Практический курс сквозного проектирования цифровых устройств на основе ПЛИС фирмы Xilinx

(часть 5)

Валерий Зотов (Москва)

Пятая часть курса знакомит с основами языка описания аппаратуры VHDL, необходимыми для подготовки описания разрабатываемого устройства. Рассмотрен синтаксис основных операторов, используемых в составе описаний проектируемых устройств и поддерживаемых средствами синтеза САПР серии Xilinx ISE.

Краткие сведения об основных элементах языка описания аппаратуры VHDL

Подробное систематизированное описание языка VHDL выходит за рамки журнальной публикации. Потому в настоящем и последующих разделах приводятся краткие сведения об основных понятиях, элементах и конструкциях этого языка, позволяющих быстро приступить к практической работе. Более подробную информацию о применении языка VHDL для описания цифровых устройств можно найти в [12–14].

Текст описания цифрового устройства на языке VHDL формируется из ключевых слов, идентификаторов, литералов и комментариев. Под ключевым (зарезервированным) словом понимается последовательность символов, которая имеет строго определённое назначение. Примером ключевых слов являются операторы. Полный список ключевых слов языка VHDL можно найти в документации или литературе [13].

Идентификаторы представляют собой последовательность символов, определяемую пользователем. Идентификаторы используются, в частности, в качестве имён сигналов, переменных, функций, процедур. В составе идентификаторов могут использоваться символы латинского алфавита, цифры и символ подчеркивания. При этом символ подчеркивания не может использоваться в качестве окончания идентификатора или следовать за аналогичным символом подчеркивания. Начинаться идентификатор может только с алфавитного символа.

В отличие от различных языков программирования, запись ключевых слов и идентификаторов в языке VHDL инвариантна по отношению к регистру символов. Таким образом, идентификаторы и ключевые слова, записанные с помощью строчных и заглавных букв, воспринимаются компилятором VHDL и средствами синтеза одинаково. Литералы, как правило, используются в качестве значений, которые присваиваются переменным или сигналам, а также для инициализации констант соответствующего типа. В языке VHDL различают следующие типы литералов: десятичные, базовые, символьные, строчные и строчные битовые. Десятичные литералы представляют собой числовые значения и подразделяются на целые и вещественные, которые могут быть представлены в обычной или экспоненциальной форме. Например, 1 – целый литерал, 7E3 – целый с экспонентой, 5.0 – вещественный, 1.7E-5 – вещественный с экспонентой.

Базовые литералы являются числовыми значениями с явным указанием системы счисления, которые представлены в следующем формате: <система_счисления>#<числовое_значение>#. Например, 2#1010# – базовый двоичный литерал, 10#25# – базовый десятичный литерал, 16#8F# – базовый шестнадцатеричный литерал. Символьные литералы образуют символы, заключенные в одиночные апострофы, например, 'C', 'Z'. Строчные литералы представляют собой последовательность алфавитных символов, заключенных в апострофы. Строчные битовые литералы содержат последовательность двоичных, восьмеричных или шестнадцатеричных символов, заключенных в апострофы, перед которыми указано условное обозначение системы представления: B – двоичная, O – восьмеричная, X – шестнадцатеричная, например, B"0011", O"5743", X"1AFC". Комментарием считается часть строки или вся строка, которая начинается с двух следующих подряд дефисов (символов –-) и заканчивается концом этой строки. Комментарии, так же как пробелы и символы перевода строки, игнорируются компилятором и средствами синтеза VHDL.

В составе языка VHDL применяется три основных класса элементов (объектов данных): сигналы, переменные и константы. Под сигналом понимается идентификатор, который соответствует некоторому физическому сигналу (физической цепи) разрабатываемого устройства. Значение, присвоенное этому идентификатору, соответствует значению физического сигнала. Кроме того, сигналы имеют ряд вспомогательных параметров, которые представлены в форме атрибутов. Атрибут сигнала указывается непосредственно после его идентификатора, отделяясь от него одиночным апострофом <идентификатор_сигнала>' <идентификатор_атрибута>.

Атрибуты бывают двух видов: предопределённые и определяемые пользователем. Наиболее часто в описании проектируемого устройства используется предопределённый атрибут *event*, который имеет тип *boolean*. Данный атрибут принимает значение *true* только в том слу-

чае, если произошло изменение этого сигнала. Например, атрибут *ADR_EN'event* будет иметь значение *true* при любом переключении сигнала *ADR_EN*. Информацию о других предопределённых атрибутах можно найти в [13].

Переменная с точки зрения разработчика представляет собой идентификатор, значение которого может изменяться различными операторами. Переменные используются в функциях и процедурах для организации внутренних вычислений в описании проектируемого устройства. Константа – идентификатор, который используется для обозначения некоторого постоянного значения. Константы могут использоваться в качестве значений переменных, параметров функций и процедур. Применение констант вместо числовых значений делает текст описания более прозрачным. Если некоторое числовое значение многократно используется в описании устройства, то, применяя вместо него соответствующую константу, в случае необходимости его изменения достаточно в одном месте поменять значение этой константы.

Для каждого из элементов, перечисленных выше, в описании устройства указывается соответствующий тип, определяющий всю совокупность значений, которые может принимать данный элемент (сигнал, переменная или константа). В языке VHDL имеется только несколько предопределённых типов, среди которых наиболее часто используются типы *bit*, *bit_vector*, *integer*, *real*, *boolean*, *character*, *string*, *time*. В частности, элементы типа *bit* могут принимать значения 0 и 1; *integer* – значения целого типа; *real* – значения вещественного типа; *boolean* – значения *true* и *false*. В отличие от предопределённых типов, остальные типы, применяемые в описании цифровых устройств, относятся к группе определяемых пользователем типов. Из таких типов в описаниях устройств чаще всего используется тип *std_logic*. Тип *std_logic* целесообразно использовать для описания большинства цифровых сигналов. Сигналы, переменные и константы этого типа могут принимать следующие значения: '0' – логический ноль, '1' – логическая единица, 'Z' – состояние высокого импеданса, 'X' – неизвестное значение, 'L' – логический ноль (слабый источник), 'H' – логическая единица (слабый источник), 'W'- неизвестное значение (слабый источник), 'U' – неинициализированное состояние, '-' – неопределённое состояние.

Кроме простых (скалярных) типов применяются составные типы данных – массивы. Массив представляет собой упорядоченную совокупность элементов одного типа, обращения к которым осуществляются с помощью индекса. Массивы в языке VHDL применяются, в первую очередь, для описания сигналов, имеющих шинную организацию. Для описания таких сигналов чаще всего используется тип массива *std_logic_vector*, элементы которого относятся к рассмотренному выше типу *std_logic*. Среди предопределённых типов, перечисленных выше, *bit_vector* также относится к типу массива.

Для осуществления основных (базовых) операций над элементами описания в языке VHDL предусмотрен набор встроенных операторов. В этот набор входят логические операторы языка *and* (логическое И), *or* (логическое ИЛИ), *not* (логическое НЕ), *nand* (логическое И-НЕ), *nor* (логическое ИЛИ-НЕ), *xor* (исключающее ИЛИ), *xnor* (исключающее ИЛИ-НЕ), операторы отношения = (равно), /= (не равно), > (больше), >= (больше или равно), < (меньше), <= (меньше или равно) и арифметические операторы + (сложения), – (вычитания), * (умножения) и / (деления). Для каждого типа элементов языка VHDL предназначена соответствующая группа встроенных операторов.

Структура описания проектируемого устройства на языке VHDL

При описании проектируемого устройства с помощью языка VHDL оно представляется в форме соответствующего объекта, который может иметь как простую (одноуровневую), так и иерархическую (многоуровневую) структуру. Исчерпывающая информация об этом объекте указывается в форме его интерфейса и архитектуры. Интерфейс объекта описывается в виде совокупности портов, каждый из которых соответствует входу, выходу или двунаправленному входу/выходу проектируемого устройства. Для каждого порта указывается его тип и режим функционирования. Архитектура объекта представляется в форме структуры разрабатываемого устройства или описания алгоритма его функционирования.

В общем случае структура VHDL-описания проектируемого устройства включает в себя три части, которые расположены в порядке их перечисления. В первой части приводятся ссылки на все используемые библиотеки, пакеты и элементы этих библиотек. Применение библиотек позволяет значительно сократить объём описания на языке VHDL. В библиотеках, в частности, содержится определение типов сигналов, переменных и функций, которые могут использоваться в различных проектах. Вторая часть представляет собой декларацию объекта описания. В этой части объявляется объект, который соответствует разрабатываемому устройству, приводится описание его интерфейса и настраиваемых параметров. В третьей части выполняется определение архитектуры описываемого объекта. Здесь приводится в той или иной форме информация, которая полностью описывает функционирование этого объекта. Далее последовательно рассматривается синтаксис конструкций языка VHDL, которые используются в каждой части описания разрабатываемого устройства.

Использование библиотек и пакетов в составе VHDL-описания проектируемого устройства

Создание ссылки на требуемую библиотеку выполняется с помощью ключевого слова library. При этом используется следующий формат записи, используемый для указания необходимых библиотек:

```
library <название_библиотеки>;
```

Кроме ссылки на библиотеку, необходимо указать конкретный пакет данной библиотеки и элементы этого пакета, которые будут использоваться в VHDL-описании проектируемого устройства. Под пакетом в языке VHDL понимается файл, содержащий декларации типов данных, переменных, констант, сигналов и компонентов, а также определения функций и процедур, которые могут применяться в различных проектах. Для указания используемого пакета библиоте-

ки и его элементов предназначено ключевое слово *use*. Формат предложения, которое открывает доступ к пакету указанной ранее библиотеки и его элементам, имеет следующий вид:

```
use<название_библиотеки>.<название_пакета>.<идентификатор_элемента_пакета>
```

Чтобы использовать все элементы пакета, указываемого в предложении *use*, необходимо использовать следующий вариант формата этого предложения:

```
use <название_библиотеки>.<название_пакета>.all
```

В качестве примера далее приводится ссылка на наиболее часто используемую библиотеку IEEE и пакет *std_logic_1164*. Определения, содержащиеся в стандартном логическом пакете *std_logic_1164* указанной библиотеки, разработанном Институтом инженеров по электротехнике и электронике, применяются в большинстве создаваемых VHDL-описаний проектируемых устройств.

```
library ieee;
use ieee.std_logic_1164. all;
```

Декларации объекта в составе VHDL-описания проектируемого устройства

В общем случае синтаксис конструкции, предназначенной для декларации описываемого объекта, выглядит следующим образом:

```
entity <имя_объекта> is
generic (
        <идентификатор_параметра1> : <тип_параметра1>;
        <идентификатор_параметра2> : <тип_параметра2>;
…
<идентификатор_параметраM> : <тип_параметраM>
);
port (
<название_порта1>: <направление_передачи_сигнала_порта1>
<тип_порта1>;
<название_порта2>: <направление_передачи_сигнала_порта2>
<тип_порта2>;
…
<название_портаN> :<направление_передачи_сигнала_портаN>
<тип_портаN>
        );
end <имя_объекта>;
```

Декларация объекта описания начинается с ключевого слова *entity*, вслед за которым указывается имя этого объекта и ключевое слово *is*. После объявления объекта в общем случае могут быть перечислены его настраиваемые параметры, которые используются в определении архитектуры разрабатываемого устройства. Применение настраиваемых параметров в VHDL-описаниях позволяет, в частности, создавать универсальные модули, которые затем при заданных значениях параметров могут использоваться в различных проектах. В форме параметров целесообразно указывать разрядность компонентов и шин, значения длительностей задержек сигналов. Например, в формируемом описании регистра (или счётчика) вместо явного указания его разрядности можно задействовать соответствующий параметр. Таким образом, впоследствии в качестве компонента можно использовать один и тот же объект для описания регистров (или счётчиков) с различной разрядностью в составе новых проектов.

Декларация настраиваемых параметров осуществляется с помощью ключевого слова *generic*, вслед за которым приводится список идентификаторов параметров с указанием их типов. Если настраиваемые параметры не применяются в составе формируемого VHDL-описания, то ключевое слово *generic* не указывается.

Далее приводится информация об интерфейсе описываемого объекта, которая начинается с ключевого слова *port*. После этого ключевого слова приводится последовательность выражений, в которых указывается название порта, режим его работы (направление передачи сигнала, ассоциированного с данным портом) и тип (соответствующий типу ассоциированного с ним сигнала). Для обозначения режима работы порта, соответствующего направлению передачи сигнала, ассоциированного с данным портом, предопределены следующие ключевые слова:

- *in* – соответствует входному порту;
- *out* – соответствует выходному порту;
- *inout* – соответствует двунаправленному (выходному/выходному) порту;
- *buffer* – соответствует выходному порту, сигнал которого можно читать в структуре описываемого объекта.

Сигналу, ассоциированному с входным портом *in* некоторого объекта, нельзя присваивать значения внутри определения архитектуры этого объекта. Значение сигнала выходного порта *out* объекта не может быть присвоено другому сигналу в составе описания архитектуры этого объекта. Сигнал, соответствующий двунаправленному порту *inout*, может использоваться в описании как входной и как выходной. Завершает декларацию объекта ключевое слово *end* с указанием названия этого объекта.

Примером декларации объекта VHDL-описания является объявление объекта *countN*, который соответствует (N+1)-разрядному двоичному счётчику с входом разрешения счёта:

```
entity countN is
generic ( N : natural );
port (
clk: in std_logic; -- порт тактового сигнала
ce: in std_logic; -- порт сигнала разрешения счёта
qout: out std_logic_VECTOR(N downto 0) -- (N+1)-разрядный порт выходного сигнала
);
end countN ;
```

Определение архитектуры объекта, представляющего разрабатываемое устройство

Определение архитектуры описываемого объекта выполняется в форме архитектурного тела (architecture body). В общем случае архитектурное тело объекта имеет следующий формат:

```
architecture <название_архитектуры> of <имя_объекта> is
[декларация используемых типов данных]
[декларация внутренних сигналов]
[декларация используемых констант]
[определения функций]
[определения процедур]
```

```
[декларация используемых компо-
нентов]
begin
<параллельный оператор1>;
<параллельный оператор2>;
…
<параллельный операторN>;
end <название_архитектуры>;
```

В квадратных скобках здесь и далее приведены необязательные элементы и параметры описания. Определение архитектуры объекта начинается с ключевого слова *architecture*, после которого указывается её название, ключевое слово *of*, имя объекта и ключевое слово *is*. В качестве названия архитектуры объекта, как правило, указывается идентификатор, производный от имени объекта. Выражения декларации типов, сигналов, переменных и констант могут располагаться в составе архитектурного тела в произвольном порядке. Далее следует ключевое слово *begin*, которое открывает блок описания структуры или поведения объекта. Последний заканчивается ключевым словом *end*, сопровождаемым названием архитектуры.

В последующих разделах приводится синтаксис конструкций языка VHDL, используемых для декларации типов данных, сигналов, констант, компонентов и определения функций и процедур. Затем рассматривается формат основных параллельных и последовательных операторов, предназначенных для описания функционирования разрабатываемого устройства.

Декларация используемых типов данных, сигналов, констант и компонентов, используемых в составе определения архитектуры VHDL-описания проектируемого устройства

Синтаксис выражения, предназначенного для объявления нового типа, который используется в определении архитектуры объекта, выглядит следующим образом:

```
type <название_типа> is (<спи-
сок_возможных_значений>);
```

Декларация нового типа, определяемого пользователем, начинается с ключевого слова *type*, после которого вводится название декларируемого типа, ключевое слово *is* и список всех возможных значений этого типа. Список возможных значений может указываться в виде перечисления их через запятую или с помощью ключевых слов *range...to*:

```
type <название_типа> is (<значе-
ние1>, [<значение2>, … ,[<значе-
ниеN>]);
type <название_типа> is range
<начальное_значение> to <конеч-
ное_значение>;
```

Например, в следующих выражениях объявляется тип *SYMBOL_HEX*, значениями которого являются шестнадцатеричные символы, и тип *DIGIT_DEC*, список значений которого образует набор десятичных цифр от 0 до 9:

```
type SYMBOL_HEX NAME is ('0',
'1', '2', '3', '4', '5', '6',
'7', '8', '9','A', 'B', 'C',
'D', 'E', 'F');
type DIGIT_DEC is range 0 to 9;
```

Если объявляемый тип является массивом, то необходимо использовать один из следующих вариантов синтаксиса декларации:

```
type <название_типа> is array
(<начальное_значение_индекса> to
<конечное_значение_индекса>) of
<тип_элементов_массива>;
type <название_типа> is array
(<начальное_значение_индекса>
downto <конечное_значение_индек-
са>) of <тип_элементов_массива>;
```

В объявлении такого типа за ключевым словом *is* следует ключевое слово *array*, после которого указывается диапазон изменения индекса массива (возрастающий или убывающий), ключевое слово *of* и обозначение типа элементов, составляющих массив. Приведённые ниже примеры наглядно демонстрируют различные варианты декларации определяемого пользователем типа, имеющего структуру массива:

```
type CPU_WORD is array (0 to 31)
of BIT;
type BUS_SIGN is array (15 down-
to 0) of std_logic;
```

Для объявления внутренних сигналов, применяемых в определении архитектуры описываемого объекта, необходимо использовать следующий формат декларации:

```
signal <идентификатор_сигнала>:
<тип_сигнала>;
```

Объявление внутреннего сигнала начинается с ключевого слова *signal*, вслед за которым указывается идентификатор этого сигнала и его тип. В одной строке может быть объявлено несколько сигналов одного типа. Примерами объявления внутренних сигналов различного типа являются следующие строки декларации:

```
signal SIG_NAME: BOOLEAN;
signal BUS_NAME: BIT_VECTOR (3
downto 0);
signal CLK_1, RESET, SEL:
STD_LOGIC;
```

Формат конструкции, используемой для декларации константы, выглядит следующим образом:

```
constant <идентификатор_констан-
ты>: <тип_ константы> := <значе-
ние_константы >;
```

Объявление константы начинается с ключевого слова *constant*, после которого указывается имя константы и через двоеточие её тип. В большинстве случаев одновременно с объявлением константы производится определение её значения, которое отделяется от обозначения типа константы символом присвоения := (двоеточием и знаком равенства). В качестве значения константы может быть указано не только числовое значение соответствующего типа, но и простое выражение, содержащее ранее объявленные константы, например:

```
constant BUS_WIDTH: integer := 31;
constant WORD_SIZE: integer :=
BUS_WIDTH - 1;
```

Для декларации компонента, используемого в структурном описании объекта, применяется конструкция, формат которой выглядит следующим образом:

```
component <название_компонента>
generic (
        <идентификатор_парамет-
ра1> : <тип_параметра1>;
```

```
        <идентификатор_парамет-
ра2> : <тип_параметра2>;
…
<идентификатор_параметраM> :
<тип_параметраM>
);
port (
<название_порта1>: <направле-
ние_передачи_сигнала_порта1>
<тип_порта1>;
<название_порта2>: <направле-
ние_передачи_сигнала_порта2>
<тип_порта2>;
…
<название_портаN> :<направле-
ние_передачи_сигнала_портаN>
<тип_портаN>
        );
end component;
```

Открывает объявление компонента ключевое слово *component*, после которого указывается имя компонента. Затем в общем случае приводится ключевое слово *generic*, в списке которого перечисляются настраиваемые параметры объявляемого компонента. Список настраиваемых параметров компонента имеет тот же синтаксис, что и в объявлении объекта описания. Далее следует описание интерфейса этого компонента, которое имеет тот же формат, что и в декларации объекта. Завершает объявление компонента сочетание ключевых слов *end component*. В качестве примера далее приводится текст декларации компонента параллельного регистра данных *REG_N* с настраиваемым количеством разрядов, которое задаётся в виде значения параметра *WIDTH*.

```
component REG_N
        generic (WIDTH: integer);
port (
IN_DAT: in
STD_LOGIC_VECTOR(WIDTH-1 downto
0);
OUT_DAT: out
STD_LOGIC_VECTOR(WIDTH-1 downto
0);
CLK: in STD_LOGIC
        );
end component;
```

Определение функций и процедур, используемых в составе архитектуры объекта VHDL-описания проектируемого устройства

Для определения функции, используемой в архитектурном теле описываемого объекта, применяется конструкция, формат которой в общем случае имеет следующий вид:

```
function <название_функции> (
<идентификатор_параметра1>: <тип_
параметра1>;
<идентификатор_параметра2>: <тип_
параметра2>;
        …
<идентификатор_параметраN>: <тип_
параметраN>
)
return <тип_возвращаемого_значе-
ния> is
[декларация локальных типов дан-
ных]
[декларация локальных констант]
[декларация локальных переменных]
[определения вложенных функций]
[определения вложенных процедур]
begin
        последовательный оператор
1;
        последовательный оператор
2;
        …
        последовательный оператор
M;
end <название_функции>;
```

Определение функции выполняется с помощью ключевого слова *function*, вслед за которым указывается её имя. После этого в скобках приводится список входных параметров определяемой функции. Для каждого из этих параметров указывается его идентификатор и соответствующий тип. Затем с помощью ключевого слова *return* задаётся тип возвращаемого значения, за которым следует ключевое слово *is*. Далее могут приводиться объявления локальных типов данных, констант и переменных, а также определения вложенных функций и процедур.

Все объявляемые локальные элементы и определяемые вложенные функции и процедуры доступны для применения только внутри определяемой функции. Завершает определение функции описание выполняемых операций (тело функции), которое приводится внутри блока *begin ... end*. Это описание, как правило, представлено совокупностью последовательных операторов, которые будут рассмотрены в следующих разделах.

В качестве примера далее приводится определение функции *INCOUNT7*, которая выполняет инкрементное изменение значения аргумента целого типа INCNT в пределах от 0 до 7:

```
function INCOUNT7 (INCNT: INTE-
GER) return INTEGER is
 variable SUM: INTEGER;
begin
 if INCNT >= 7 then
 SUM := 0;
 else
 SUM : INCNT + 1;
 end if;
 return SUM;
end INCOUNT7;
```

Процедуры языка VHDL по своему назначению подобны функциям. Отличия заключаются в том, что процедуры не имеют возвращаемых значений, но при этом, кроме входных аргументов, имеют ещё выходные и входные/выходные параметры. Поэтому при вызове функции её название указывается в правой части выражения присваивания, а вызов процедур аналогичен использованию операторов. Определение процедуры осуществляется с помощью конструкции, формат которой выглядит следующим образом:

```
procedure <название_процедуры>
(
<класс_ параметра1> <идентифика-
тор_параметра1>: <режим_использо-
вания_параметра1> <тип_ парамет-
ра1>;
<класс_ параметра2> <идентифика-
тор_параметра2>: <режим_использо-
вания_параметра2> <тип_ парамет-
ра2>;
        …
<класс_ параметраN> <идентифика-
тор_параметраN>: <режим_использо-
вания_параметраN> <тип_ парамет-
раN>
)
is
[декларация локальных типов дан-
ных]
[декларация локальных констант]
[декларация локальных переменных]
[определения вложенных функций]
[определения вложенных процедур]
begin
-- содержание процедуры
        последовательный оператор
1;
        последовательный оператор
2;
        …
```

```
        последовательный оператор
M;
end <название_ процедуры>;
```

В начале определения процедуры указывается ключевое слово *procedure* и её название. Затем в скобках приводится список всех параметров определяемой процедуры. Для каждого из этих параметров указывается обозначение класса, к которому он относится, его идентификатор, режим использования и соответствующее название типа данных. В качестве параметров процедуры могут использоваться переменные, константы и сигналы, поэтому перед идентификатором параметра приводится наименование класса, к которому он относится.

Режим использования параметра, указывающий направление передачи данных, определяется с помощью ключевых слов *in*, *out*, *inout*, которые соответствуют входному, выходному и входному/выходному параметру. За списком параметров следует ключевое слово *is*, после которого может располагаться блок объявления локальных типов данных, констант, переменных и определения вложенных функций и процедур, предназначенных для применения только внутри определяемой процедуры. Совокупность операций, выполняемых определяемой процедурой (содержание процедуры), описывается внутри блока *begin ... end*. В качестве примера ниже приведён текст определения процедуры маскирования восьмиразрядного сигнала:

```
procedure MASK (
signal XIN: in STD_LOGIC_VECTOR;
signal XOUT: out STD_LOGIC_VEC-
TOR
)
is
constant MSK: STD_LOGIC_VECTOR;
variable TMP: STD_LOGIC_VECTOR;
begin
for I in 0 to 7 loop
TMP(I) := XIN(I) and MSK(I);
end loop;
XOUT <= TMP;
end;
```

Параллельно выполняемые операторы языка VHDL

В отличие от обычных (последовательно выполняемых) операторов, используемых в других языках высокого уровня, в языке VHDL присутствуют операторы, которые выполняются параллельно. Применение таких операторов позволяет наиболее достоверно описать функционирование цифрового устройства. Порядок следования параллельных операторов в определении архитектуры описываемого объекта не оказывает влияния на описание его функционирования. В группу параллельно выполняемых операторов языка VHDL, которые наиболее часто используются для описания разрабатываемых устройств, входят:

- оператор процесса;
- параллельный оператор присваивания значения сигнала;
- оператор условного присваивания значения сигнала;
- оператор выборочного присваивания значения сигнала;
- оператор создания экземпляра (конкретизации) компонента;
- оператор генерации.

Под процессом в языке VHDL понимается совокупность последовательных операторов, которая выполняется «одновременно» с другими параллельными операторами при изменении одного из сигналов, связанных с этим процессом. Процесс может находиться в одном из следующих состояний: в состоянии выполнения, ожидания или приостановленном состоянии. Совокупность сигналов, оказывающих влияние на изменение состояния процесса, называют списком чувствительности данного процесса. Формат оператора процесса в общем случае имеет следующий вид:

```
[<метка>]: process (<идентифика-
тор_сигнала1>, <идентифика-
тор_сигнала2>, …,<идентифика-
тор_сигналаN>)
[декларация локальных типов дан-
ных]
[декларация локальных констант]
[декларация локальных переменных]
[определения вложенных функций]
[определения вложенных процедур]
begin
-- содержание процесса
        последовательный оператор
1;
        последовательный оператор
2;
        …
        последовательный оператор
M;
end process [<метка>];
```

Началом описания процесса является ключевое слово *process*, после которого в скобках указывается список чувствительности этого процесса. В ряде случаев этот список может отсутствовать. Перед ключевым словом *process* можно задать необязательную метку, которая может использоваться в качестве идентификатора этого процесса. В дальнейшем если метка носит необязательный характер, то она не будет упоминаться при рассмотрении синтаксиса операторов.

После списка чувствительности может располагаться раздел, содержащий объявления типов данных, констант и переменных, а также определения функций и процедур, которые предназначены для использования только внутри этого процесса. Этот раздел не является обязательным и довольно часто отсутствует в описании процесса. Содержание процесса в виде совокупности последовательных операторов приводится после ключевого слова *begin*. В теле процесса могут присутствовать операторы ожидания *wait*, различные формы которого будут рассмотрены в следующем разделе. Завершает оператор процесса сочетание ключевых слов *end process*. Использование оператора процесса иллюстрирует следующий пример:

```
process (CLK)
begin
        SIGN_D0 <= Data_In_A &
Data_In_B;
        SIGN_D1 <= Data_In_A &
Data_In_C;
        if (SIGN_En = '0') then
                Data_OUT <=
SIGN_D0 xor SIGN_D1;
        else
                Data_OUT <=
SIGN_D0 or SIGN_D1;
end process;
```

Для присвоения значений сигналам при определении архитектуры описываемого объекта применяются параллельные операторы безусловного, условного и выборочного назначения сигнала. Формат параллельного оператора безусловного присваивания значения сигнала имеет следующий вид:

```
<идентификатор_сигнала> <=
[<вид_задержки>] <выражение,_оп-
```

```
ределяющее_значение_сигнала>
[after] [<длительность_задерж-
ки>];
```

В левой части этого оператора указывается имя сигнала, которому присваивается значение. В правой части этого оператора может быть указано некоторое конкретное значение, соответствующее типу сигнала, или выражение, которое определяет алгоритм формирования сигнала. Результат этого выражения должен иметь тот же тип, что и сигнал. В операторе присвоения значения сигналу можно также указать задержку, с которой происходит изменение состояние этого сигнала. Для этого используется ключевое слово *after*, которое должно располагаться после присваиваемого значения сигнала. При этом перед выражением, определяющим значение сигнала, можно указать вид задержки с помощью соответствующего ключевого слова.

В языке VHDL используется два вида задержек: инерционный и транспортный, которые обозначаются ключевыми словами *inertial* и *transpor* соответственно. При инерционной задержке сигнал изменяет своё состояние только в том случае, если длительность нового состояния (присваиваемого значения) превышает указанное значение задержки. В случае транспортной задержки сигнал изменяет своё значение при любом соотношении длительности этого состояния и величины задержки. Если не указано ключевое слово, определяющее вид задержки, то по умолчанию задержка считается инерционной. Приведённые ниже примеры демонстрируют различные формы использования параллельного оператора назначения сигнала:

```
ADR_SIGN <= N1 and N2;
SIGN_SET <= '1' after 100 ns;
```

При использовании параллельного оператора условного назначения сигнала определённое значение присваивается сигналу только при выполнении соответствующего условия. В одном операторе может быть указано несколько значений и соответствующих им условий. Если не выполнено первое условие, то проверяется выполнение второго и т.д., пока не будет обнаружено выполненное условие. Если ни одно из указанных условий не выполнено, то сигналу присваивается значение, которое следует за последним ключевым словом *else*.

Для записи условных выражений используются встроенные логические операторы языка VHDL и операторы отношения. Следует обратить внимание на то, что синтаксис оператора отношения «меньше или равно» (<=) аналогичен формату оператора присвоения значения сигнала. Назначение этого оператора определяется контекстом, в котором он используется. Формат параллельного оператора условного назначения сигнала выглядит следующим образом:

```
<идентификатор_сигнала> <= <выра-
жение1,_определяющее_значе-
ние_сигнала> [after] [<длитель-
ность_задержки1>] when <услов-
ное_выражение1> else
<выражение2,_определяющее_значе-
ние_сигнала> [after] [<длитель-
ность_задержки2>] when <услов-
ное_выражение2> else
…
<выражениеN-1,_определяющее_зна-
чение_сигнала> [after] [<длитель-
ность_задержкиN-1>] when <услов-
ное_выражениеN-1> else
<выражениеN,_определяющее_значе-
ние_сигнала> [after] [<длитель-
ность_задержкиN>];
```

В правой части этого оператора приводится значение или выражение, которое определяет алгоритм формирования сигнала, после которого с помощью ключевого слова *when* указывается соответствующее условие. Далее приводится ключевое слово *else*, за которым указываются альтернативные условия и соответствующие значения. При необходимости можно указать задержки изменения значения сигнала в той же форме, что и в предыдущем операторе. Примером условного назначения сигнала является выражение:

```
SIGN1 <= '1' when SIGN2 = '1'
and SIGN3 = '0' else '0';
```

Параллельный оператор выборочного назначения сигнала позволяет присвоить одно из списка возможных значений в зависимости от значения некоторого выражения (выражения выбора). Этот оператор имеет следующий формат:

```
with <выражение_выбора> select
<идентификатор_сигнала> <= <выра-
жение1,_определяющее_значе-
ние_сигнала> [after] [<длитель-
ность_задержки1>] when <значе-
ние1_выражения_выбора>,
<выражение2,_определяющее_значе-
ние_сигнала> [after] [<длитель-
ность_задержки2>] when <значе-
ние2_выражения_выбора>,
…
<выражениеN-1,_определяющее_зна-
чение_сигнала> [after] [<длитель-
ность_задержкиN-1>] when
<значениеN-1_выражения_выбора >,
<выражениеN,_определяющее_значе-
ние_сигнала> [after] [<длитель-
ность_задержкиN>] when others;
```

Началом оператора избирательного назначения сигнала является ключевое слово *with*, после которого указывается выражение выбора. Значение этого выражения выбора определяет соответствующее значение сигнала. Затем следует ключевое слово *select* и идентификатор сигнала. Далее приводятся возможные присваиваемые значения сигнала и соответствующие им значения выражения выбора, разделённые ключевым словом *when*. Необязательная информация о задержках переключения сигнала указывается в той же форме, что и в предыдущих операторах. С помощью сочетания ключевых слов *when others* в конце оператора указывается значение сигнала, которое присваивается в том случае, когда выражение выбора принимает значение, отличающееся от всех перечисленных выше. Например,

```
with BIN_VAL select
        SIG_V <=        "01111001"
when «0001",
                        "00100100"
when «0010",
                        "00110000"
when «0011",
                        "00011001"
when «0100",
                        "00010010"
when «0101",
                        "01000000"
when others;
```

Оператор конкретизации компонента выполняет функцию создания экземпляра этого компонента и определения связей с другими компонентами в структурном описании

объекта проекта. В качестве компонентов могут использоваться определённые ранее функциональные блоки разрабатываемого устройства или библиотечные элементы. Каждый компонент, используемый при определении архитектуры объекта, должен быть объявлен в блоке декларации этой архитектуры. В составе описания объекта проектируемого устройства может использоваться несколько экземпляров одного компонента. Чтобы их различать, каждому экземпляру компонента присваивается оригинальная метка, которая соответствует позиционному обозначению элемента на принципиальной схеме. Синтаксис оператора конкретизации компонента в общем случае выглядит следующим образом:

```
<метка>: <название_компонента>
port map(список_соответствия_портов_и_сигналов);
```

При формировании конкретного экземпляра компонента указывается позиционная метка – название компонента, которое определяется при создании его описания, и с помощью сочетания ключевых слов *port map* определяется список соответствия портов и сигналов (цепей). Этот список может быть представлен в виде позиционного или ключевого сопоставления портов экземпляра компонента и подключаемых сигналов. Поэтому на практике применяется два формата данного оператора. При позиционном сопоставлении подключаемые сигналы в списке перечисляются в том же порядке, что и порты в описании используемого компонента. В этом случае применяется следующий формат оператора конкретизации компонента:

```
<метка>: <название_компонента>
port map (
                <идентификатор_сигнала1>,
<идентификатор_сигнала2>,
                …
<идентификатор_сигналаN>
);
```

При ключевом сопоставлении портов создаваемого экземпляра компонента и подключаемых сигналов порядок их перечисления не имеет значения. Для каждого порта компонента в явной форме указывается идентификатор соответствующего сигнала. Таким образом, может применяться ещё один вариант формата оператора конкретизации компонента:

```
<метка>: <название_компонента>
port map (
                <название_порта1>
=> <идентификатор_сигнала1>,
<название_порта2> => <идентификатор_сигнала2>,
                …
<название_портаN> => <идентификатор_сигналаN>
);
```

В качестве примера ниже приведены операторы создания двух экземпляров компонента *COMP_DIG*, которые демонстрируют применение двух форматов этого оператора:

```
D1: COMP_DIG
        port map(INSIGN_A,
INSIGN_B, OUTSIGN);
D2: COMP_DIG
        port map(IPORT_A =>
INSIGN_A,
IPORT_B => INSIGN_B,
OPORT_B => OUTSIGN);
```

Для компактного описания регулярных структур предназначен оператор генерации *generate*, который позволяет автоматически формировать повторяющиеся блоки описания архитектуры объекта. Данный оператор может применяться в двух вариантах: циклическом и условном. Циклический вариант оператора генерации имеет следующий формат:

```
<метка>: for <идентификатор_параметра_генерации> in <начальное_значение_ параметра_генерации> to <конечное_значение_параметра_генерации> generate
<параллельный оператор1>;
<параллельный оператор2>;
…
<параллельный операторN>;
end generate;
```

В начале оператора указывается метка, за которой следует ключевое слово *for*, и идентификатор параметра генерации, диапазон изменения которого определяется с помощью ключевых слов *in* и *to*. Затем указывается ключевое слово *generate*, после которого приводится список параллельных операторов, определяющих содержимое формируемых блоков. Завершает оператор сочетание ключевых слов *end generate*. Параметр генерации аналогичен параметру цикла в различных языках программирования высокого уровня.

Синтаксис условного варианта оператора генерации выглядит следующим образом:

```
<метка>: if <условное_выражение>
generate
<параллельный оператор1>;
<параллельный оператор2>;
…
<параллельный операторN>;
end generate;
```

В этом варианте после метки указывается ключевое слово *if* и условие, при котором будет выполняться генерация формируемых блоков. Содержание этих блоков определяет совокупность параллельных операторов, указанных после ключевого слова *generate*. Условное выражение в этом операторе имеет тот же формат, что и в условном операторе назначения сигнала. В качестве примера ниже приведены два варианта использования оператора генерации для формирования блока, выполняющего операции инверсии и поразрядного логического И для пятиразрядных шин:

```
G1: for J in 0 to 4 generate
                D1: INV port map
(

I => IN_DAT(J),
O => OUT_DAT(J) );
end generate;
G2: if L >= 0 and L<= 4 generate
D1: AND2 port map (
                                I0
=> IN_ADAT(J),
I1 => IN_BDAT(J),
O => OUT_ANDAT(J) );
end generate;
```

Продолжение следует

Литература

12. *Бибило П.Н.* Основы языка VHDL. Солон-Р, 2000.
13. *Бибило П.Н.* Синтез логических схем с использованием языка VHDL. Солон-Р, 2002.
14. *Уэйкерли Дж.Ф.* Проектирование цифровых устройств. Том 1. Постмаркет, 2002.