Продолжение. Начало в № 2`2010

Разработка VHDL-описаний цифровых устройств, проектируемых на основе ПЛИС фирмы Xilinx, с использованием шаблонов САПР ISE Design Suite

Вторая часть статьи завершает описание шаблонов наиболее часто используемых базовых конструкций языка VHDL, расположенных в папке Common Constructs. В этой части рассматриваются шаблоны функций преобразования типов данных, создания ссылок на требуемые библиотеки и используемые пакеты этих библиотек, основных операторов, выражений декларации интерфейсных портов, предопределенных атрибутов, а также конструкций, предназначенных для определения функций и процедур и их вызова.

Валерий ЗОТОВ
walerry@km.ru

Шаблоны функций преобразования типов данных (окончание)

Раздел *IEEE_STD_LOGIC_ARITH*, представленный в папке *Conversion Functions* (рис. 6, см. КиТ № 2`2010), содержит шаблоны функций преобразования, которые относятся к следующим типам данных: SIGNED, UNSIGNED, SMALL_INT, INTEGER, STD_ULOGIC, STD_LOGIC и STD_LOGIC_VECTOR. Эти функции определены в пакете *std_logic_arith* библиотеки IEEE [14].

INTEGER to INTEGER — это шаблон выражения, используемый для обращения к функции CONV_INTEGER, которая предназначена для приведения данных различных типов к целому типу INTEGER.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<int_sig> = CONV_INTEGER(<int_sig>);
```

В этом варианте применения функции CONV_INTEGER в качестве ее аргумента <int_sig> могут использоваться константа, переменная или сигнал целого типа.

INTEGER to SIGNED представляет собой шаблон варианта использования функции CONV_SIGNED, который предназначен для приведения данных целого типа INTEGER к знаковому типу SIGNED.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<signed_sig> = CONV_SIGNED(<int_sig>, <integer_size>);
```

Аргументом функции CONV_SIGNED в представленном варианте является преобразуемое значение константы, переменной или сигнала целого типа INTEGER. Целочисленный параметр <integer_size> определяет разрядность значения результата преобразования, выраженную в количестве бит. Идентификатор преобразуемого значения константы, переменной или сигнала указывается вместо <int_sig>. Значение рассматриваемой функции может быть присвоено сигналу или переменной знакового типа SIGNED, идентификатор которых заменяет <signed_sig>.

INTEGER to STD_LOGIC_VECTOR включает в себя шаблон варианта применения функции CONV_STD_LOGIC_VECTOR для преобразования значения переменной или сигнала целого типа INTEGER в массив типа STD_LOGIC_VECTOR.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<slv_sig> = CONV_STD_LOGIC_VECTOR(<int_sig>, <integer_size>);
```

В рассматриваемом варианте функция CONV_STD_LOGIC_VECTOR имеет тот же аргумент, что и предыдущая функция преобразования типа. При этом параметр <integer_size>, задаваемый в виде константы, переменной или значения целого типа, определяет длину диапазона индекса массива типа STD_LOGIC_VECTOR, формируемого в результате преобразования. В случае использования данного шаблона в составе VHDL-описания разрабатываемого устройства следует в последнем выражении заменить <slv_sig> именем массива указанного типа.

INTEGER to UNSIGNED содержит образец варианта использования функции CONV_UNSIGNED, который выполняет приведение данных целого типа INTEGER к беззнаковому типу UNSIGNED.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<unsigned_sig> = CONV_UNSIGNED(<int_sig>, <integer_size>);
```

В качестве аргумента функции CONV_UNSIGNED <int_sig> в этом варианте указывается идентификатор константы, переменной или сигнала целого типа INTEGER. Параметр, определяющий разрядность значения результата преобразования, задается в виде константы, переменной или значения целого типа. Соответствующий идентификатор или целочисленное значение заменяет <integer_size> в выражении вызова рассматриваемой функции. Значение функции CONV_UNSIGNED присваивается переменной или сигналу, которые относятся к беззнаковому типу UNSIGNED. Идентификатор этого сигнала или переменной записывается вместо <unsigned_sig>.

SIGNED to INTEGER включает в себя шаблон варианта использования функции TO_INTEGER для преобразования данных знакового типа SIGNED в данные целого типа INTEGER.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<int_sig> = CONV_INTEGER(<signed_sig>);
```

В качестве аргумента функции CONV_INTEGER в данном случае нужно указать идентификатор константы, переменной или сигнала знакового типа (вместо <signed_sig>). Значение функции TO_INTEGER может быть присвоено переменной или сигналу целого типа, идентификатор которого заменяет <int_sig>.

SIGNED to SIGNED — это образец варианта применения функции CONV_SIGNED для конверсии данных знакового типа в тот же тип, но с другой разрядностью.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<signed_sig> = CONV_SIGNED(<signed_sig>, <integer_size>);
```

В этом варианте аргументом функции CONV_SIGNED является константа, переменная или сигнал знакового типа SIGNED, идентификатор которых следует указать вместо <signed_sig>. Параметр, определяющий разрядность результата преобразования, задается в виде константы, переменной или значения целочисленного типа INTEGER, идентификатор которых заменяет в последней строке шаблона <integer_size>. Значение функции CONV_SIGNED в рассматриваемом шаблоне присваивается переменной или сигналу знакового типа SIGNED с указанной разрядностью, идентификатор которого записывается в последнем выражении вместо <signed_sig>.

SIGNED to STD_LOGIC_VECTOR представляет собой шаблон варианта применения функции CONV_STD_LOGIC_VECTOR для преобразования значения константы, переменной или сигнала знакового типа SIGNED в массив типа STD_LOGIC_VECTOR.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<slv_sig> = CONV_STD_LOGIC_VECTOR(<signed_sig>, <integer_size>);
```

В рассматриваемом варианте использования функции CONV_STD_LOGIC_VECTOR тип и назначение аргумента совпадают с предыдущей функцией конверсии типа SIGNED to SIGNED. Параметр <integer_size> в этом случае определяет длину диапазона изменения индекса для массива, формируемого в результате преобразования. В последнем выражении шаблона, в котором непосредственно осуществляется обращение к функции CONV_STD_LOGIC_VECTOR, нужно заменить <slv_sig> названием соответствующего массива типа STD_LOGIC VECTOR.

SIGNED to UNSIGNED содержит шаблон варианта использования функции CONV_UNSIGNED для приведения значения константы, переменной или сигнала знакового типа SIGNED к значению без знака UNSIGNED.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<unsigned_sig> = CONV_UNSIGNED(<signed_sig>, <integer_size>);
```

В данном случае аргументом функции CONV_UNSIGNED является константа, переменная или сигнал знакового типа SIGNED, идентификатор которого указывается вместо <signed_sig>. Разрядность результата преобразования устанавливается с помощью параметра <integer_size>, значение которого задается в виде идентификатора константы, переменной или значения целочисленного типа INTEGER. Значение рассматриваемой функции может быть присвоено сигналу или переменной беззнакового типа UNSIGNED, идентификатор которой записывается вместо <unsigned_sig>.

STD_LOGIC_VECTOR to STD_LOGIC_VECTOR (sign extend) включает в себя образец применения функции SXT, выполняющей преобразование массива типа STD_LOGIC_VECTOR в массив этого же типа заданного размера с заполнением дополнительных позиций значением старшего (знакового) разряда исходного операнда.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<slv_sxt_sig> = SXT(<slv_sig>, <integer_size>);
```

В качестве аргумента функции SXT <slv_sig> задается название массива, относящегося к типу STD_LOGIC_VECTOR. Целочисленный параметр <integer_size> определяет длину диапазона изменения индекса массива, получаемого в результате преобразования. Значение рассматриваемой функции может быть присвоено массиву типа STD_LOGIC VECTOR, имя которого указывается вместо <slv_sxt_sig>.

STD_LOGIC_VECTOR to STD_LOGIC_VECTOR (zero extend) предоставляет образец использования функции EXT, которая осуществляет трансформацию массива типа STD_LOGIC_VECTOR в массив этого же типа указанного размера с заполнением дополнительных позиций (слева) нулевым значением.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<slv_ext_sig> = EXT(<slv_sig>, <integer_size>);
```

В функции EXT используется аргумент и параметр того же типа и назначения, что и в предыдущей функции преобразования.

STD_ULOGIC to SIGNED содержит шаблон варианта применения функции CONV_SIGNED для приведения значения операнда, относящегося к типу STD_ULOGIC, к значению знакового типа SIGNED.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<signed_sig> = CONV_SIGNED(<sul_sig>, <integer_size>);
```

В последнем выражении, в котором осуществляется обращение к функции CONV_SIGNED, нужно заменить <sul_sig> идентификатором константы, переменной или сигнала, которые соответствуют типу STD_ULOGIC, а <integer_size> — именем переменной, константы или значением целого типа. Последний параметр определяет количество разрядов в значении, получаемом в результате выполнения рассматриваемой функции.

STD_ULOGIC to SMALL_INT представляет собой шаблон варианта использования функции CONV_INTEGER для преобразования значения данных, относящихся к типу STD_ULOGIC, в значение типа SMALL_INT, который, в свою очередь, является подтипом базового типа INTEGER.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<int_sig> = CONV_INTEGER(<sul_sig>);
```

В этом варианте обращения к функции CONV_INTEGER в качестве аргумента <sul_sig> должен быть указан идентификатор константы, переменной или сигнала, которые относятся к типу STD_ULOGIC. Результат преобразования присваивается переменной или сигналу, имя которых заменяет <int_sig>.

STD_ULOGIC to STD_LOGIC_VECTOR включает в себя образец применения функции CONV_STD_LOGIC_VECTOR для формирования массива типа STD_LOGIC_VECTOR из значения операнда, который принадлежат к типу STD_ULOGIC.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<slv_sig> = CONV_STD_LOGIC_VECTOR(<sul_sig>, <integer_size>);
```

В данном случае аргументом функции CONV_STD_LOGIC_VECTOR является значение константы, переменной или сигнала, которые относятся к типу STD_ULOGIC. Идентификатор соответствующего операнда заменяет в выражении обращения к функции <sul_sig>. Значение этой функции может быть присвоено массиву типа STD_LOGIC VECTOR, имя которого указывается вместо <slv_sig>. Длина диапазона изменения индекса этого массива определяется значением параметра <integer_size>, которое задается в виде идентификатора константы, переменной или значения целого типа INTEGER.

STD_ULOGIC to UNSIGNED содержит образец варианта обращения к функции CONV_UNSIGNED с целью преобразования значения операнда, который имеет тип STD_ULOGIC, в значение беззнакового типа UNSIGNED.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<unsigned_sig> = CONV_UNSIGNED(<sul_sig>, <integer_size>);
```

В этом шаблоне в качестве аргумента функции CONV_UNSIGNED <sul_sig> долж-

но быть указано имя константы, переменной или сигнала, которые относятся к типу STD_ULOGIC. Значение целочисленного параметра <integer_size> устанавливает разрядность результата преобразования. Этот параметр может быть представлен идентификатором константы, переменной или значением целого типа INTEGER.

UNSIGNED to INTEGER представляет собой шаблон варианта использования функции TO_INTEGER для приведения операндов беззнакового типа UNSIGNED к целому типу INTEGER.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<int_sig> = CONV_INTEGER(<unsigned_sig>);
```

При использовании этого шаблона следует в выражении вызова функции CONV_INTEGER заменить <unsigned_sig> идентификатором константы, переменной или сигнала, которые относятся к беззнаковому типу UNSIGNED.

UNSIGNED to SIGNED предоставляет шаблон варианта применения функции CONV_SIGNED для преобразования значений данных беззнакового типа UNSIGNED в значения со знаком SIGNED.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<signed_sig> = CONV_SIGNED(<unsigned_sig>, <integer_size>);
```

В этом варианте использования функции CONV_SIGNED ее аргумент задается в виде константы, переменной или сигнала, относящегося к беззнаковому типу UNSIGNED. В выражении обращения к рассматриваемой функции нужно вместо <unsigned_sig> указать соответствующий идентификатор операнда без знака. Параметр <integer_size>, значение которого устанавливает разрядность результата преобразования, может быть задан в форме константы или переменной целого типа.

UNSIGNED to STD_LOGIC_VECTOR включает в себя образец варианта применения функции CONV_STD_LOGIC_VECTOR для конвертирования значения константы, переменной или сигнала беззнакового типа UNSIGNED в массив типа STD_LOGIC_VECTOR.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<slv_sig> = CONV_STD_LOGIC_VECTOR(<unsigned_sig>,
<integer_size>);
```

В представленном варианте аргументом функции CONV_STD_LOGIC_VECTOR может являться значение константы, переменной или сигнала беззнакового типа UNSIGNED, идентификатор которых указывается вместо <unsigned_sig>. Параметр <integer_size>, определяющий длину диапазона изменения индекса для формируемого массива типа STD_LOGIC_VECTOR, может задаваться в виде константы, переменной или значения целого типа.

UNSIGNED to UNSIGNED содержит шаблон варианта обращения к функции CONV_UNSIGNED для приведения операндов беззнакового типа UNSIGNED к значению этого же типа, но с другой разрядностью.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_arith.all;
--
<unsigned_sig> = CONV_UNSIGNED(<unsigned_sig>, <integer_
size>);
```

В этом шаблоне в качестве аргумента функции CONV_UNSIGNED используется константа, переменная или сигнал, которые принадлежат к беззнаковому типу UNSIGNED. Соответствующий идентификатор следует указать в строке обращения к рассматриваемой функции вместо <unsigned_sig>. Разрядность результата преобразования определяется с помощью целочисленного параметра <integer_size>, который может быть задан в форме константы, переменной или значения целого типа.

В разделе *IEEE-STD_LOGIC_SIGNED*, входящем в состав папки *Conversion Functions*, представлен шаблон вызова функции преобразования массивов с учетом знака, которая применяется по отношению к данным типа STD_LOGIC_VECTOR. Определение этой функций приведено в пакете *std_logic_signed* библиотеки IEEE [14].

STD_LOGIC VECTOR to INTEGER включает в себя набор выражений, применяемых при обращении к функции CONV_INTEGER, определенной в пакете *std_logic_signed*, которая предназначена для преобразования массива типа STD_LOGIC VECTOR в значение целого типа INTEGER с учетом знака.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_signed.all;
--
<int_sig> = CONV_INTEGER(<slv_sig>);
```

При использовании этого шаблона следует в выражении вызова функции CONV_INTEGER заменить аргумент <slv_sig> именем массива типа STD_LOGIC VECTOR, а <int_sig> — идентификатором переменной или сигнала целого типа.

Раздел IEEE-STD_LOGIC_UNSIGNED, представленный в папке *Conversion Functions*, содержит шаблон функции преобразования массивов без учета знака, которая используется по отношению к операндам типа STD_LOGIC_VECTOR. Определение этой функции приведено в пакете *std_logic_unsigned* библиотеки IEEE.

STD_LOGIC VECTOR to INTEGER представляет собой образец использования функции CONV_INTEGER, которая определена в пакете *std_logic_unsigned*, для преобразования данных типа STD_LOGIC_VECTOR в целочисленное значение без учета знака.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_signed.all;
--
<int_sig> = CONV_INTEGER(<slv_sig>);
```

В качестве аргумента функции CONV_INTEGER используется массив типа STD_LOGIC VECTOR, имя которого следует указать в последней строке этого шаблона вместо <slv_sig>. Результат преобразования может быть присвоен сигналу или переменной целого типа INTEGER, идентификатор которой задается вместо <int_sig>.

Кроме рассмотренных выше шаблонов, в папке *Conversion Functions* представлен также модуль описания ***Info***, который содержит наглядные примеры использования функций преобразования типов данных, относящихся к различным группам. Текст этого модуля выглядит следующим образом.

```
-- How To Use Conversion Functions
-- ==============================
--
-- The VHDL langauge allows the usage of conversion functions. These
conversion
-- functions are generally used to convert from one type to another.
-- There are multiple conversion functions provided in different
libraries.
-- This is why it is important to reference the correct library in which the
-- conversion function is available
--
-- To make it easier a couple of examples of using conversion functions are
-- provided below.
--
-- Example of converting integer to standard logic vector(conv_std_
logic_vector):
--
-- library declaration for the conversion function
   library IEEE;
   use IEEE.std_logic_arith.all;
--
-- inside the architecture block
   signal u1 : unsigned (3 downto 0);
   ignal s1 : signed (7 downto 0);
   signal v1, v2 : std_logic_vector (3 downto 0);
   signal v3, v4 : std_logic_vector (7 downto 0);
   signal i1, i2 : integer;
   u1 <= "1101";
   s1 <= "1101";
   i1 <= 13;
   i2 <= -2;
   wait for 10 ns;
   v1 <= conv_std_logic_vector(u1, 4); -- = "1101", 4 indicates the
converted result is a 4-bit value
   v2 <= conv_std_logic_vector(s1, 4); -- = "1101", 4 indicates the
converted result is a 4 bit value
   v3 <= conv_std_logic_vector(i1, 8); -- = "1101", 8 indicates the
converted result is a 8-bit value
   v4 <= conv_std_logic_vector(i2, 8); -- = "1101", 8 indicates the
converted result is a 8-bit value
--
-- Example of converting standard logic vector to integer (Example 1):
--
-- library declaration for the conversion function
   library IEEE;
   use IEEE.std_logic_signed.all
--
-- inside the architecture block
   signal b : std_logic;
   ignal u1 : unsigned (3 downto 0);
   signal s1 : signed (3 downto 0);
   signal i1, i2 : integer;
   u1 <= "1001";
   s1 <= "1001";
   b <= 'X';
   wait for 10 ns;
```

```
    i1 <= conv_integer(u1); -- 9
    i2 <= conv_integer(s1); -- -7
--
-- Example of converting standard logic vector to integer (Example 2):
--
-- library declaration for the conversion function
    library IEEE;
    use IEEE.std_logic_signed.all
--
-- inside the architecture block
    type ram_type is array ((2**FIFO_DEPTH)-1 downto 0) of std_
logic_vector (FIFO_WIDTH-1 downto 0);
    signal fifo_ram : ram_type;
    signal rd_addr: std_logic_vector((FIFO_DEPTH-1 downto 0);
--
    process (rd_clk)
        begin
            if (rd_clk'event and rd_clk = '1') then
                dout <= fifo_ram(conv_integer(rd_addr));
            end if;
    end process;
```

Рассматриваемый шаблон включает в себя примеры использования функций конвертирования сигналов знакового и беззнакового типов в массив типа STD_LOGIC VECTOR, в также преобразования массива типа STD_LOGIC VECTOR в целочисленное значение.

Шаблоны выражений, предназначенные для создания ссылок на требуемые библиотеки и используемые пакеты этих библиотек

В разделе *Library Declarations/Use* представлены шаблоны конструкций, предназначенные для создания ссылок на требуемые библиотеки и используемые пакеты этих библиотек. Подробная структура этого раздела показана на рис. 7.

Commonly Used содержит совокупность выражений, с помощью которых создаются ссылки на библиотеку IEEE и на наиболее часто используемые пакеты этой библиотеки. Текст данного шаблона включается в состав формируемого VHDL-описания без каких-либо изменений.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
```

Подраздел IEEE включает в себя группу шаблонов, позволяющих выборочно сформировать ссылки на одноименную библиотеку и различные ее пакеты, в соответствии с требованиями разработчика.

Declaration представляет собой выражение декларации стандартной библиотеки IEEE:

```
library IEEE;
```

В папке *Packages* находятся шаблоны, предназначенные для создания ссылок на конкретные пакеты стандартной библиотеки IEEE [14–16]. Для использования требуемого пакета в создаваемом VHDL-описании достаточно включить в состав формируемого модуля соответствующий шаблон из этой папки.

math_complex содержит выражение, открывающее возможность использования всех типов данных и функций, определения

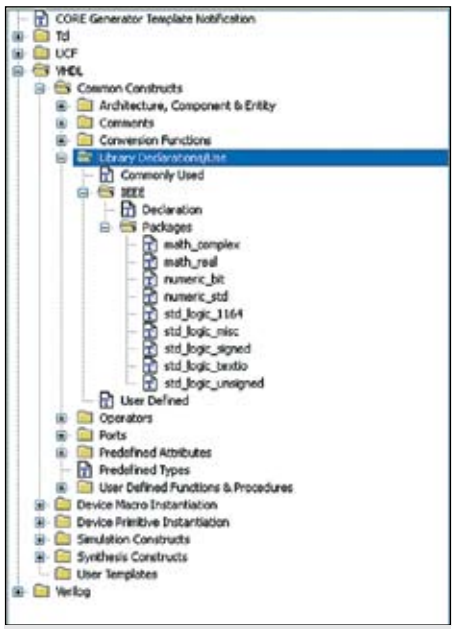

Рис. 7. Структура раздела Library Declarations/Use шаблонов базовых конструкций языка VHDL

которых представлены в составе пакета *math_complex* библиотеки IEEE. Этот пакет предназначен, в первую очередь, для осуществления математических вычислений при моделировании проектируемых устройств с использованием комплексных типов данных и соответствующих функций:

```
use IEEE.math_complex.all;
```

math_real является шаблоном выражения, которое позволяет использовать все типы данных и функции, определенные в составе пакета *math_real* библиотеки IEEE. Указанный пакет, как правило, применяется для осуществления математических вычислений с использованием данных с плавающей запятой в описаниях моделей проектируемых устройств:

```
use IEEE.math_real.all;
```

numeric_bit представляет собой выражение, предназначенное для включения в состав описания проектируемого устройства одноименного пакета стандартной библиотеки IEEE, который содержит определения арифметических, логических функций и функций сдвига, выполняемых над данными битового типа BIT:

```
use IEEE.numeric_bit.all;
```

numeric_std включает в себя выражение использования всех типов данных и функций одноименного пакета стандартной библиотеки IEEE в составе разрабатываемого VHDL-описания. В этом пакете представлены определения арифметических, логических функций и функций сдвига для данных базового типа STD_ULOGIC:

```
use IEEE.numeric_std.all;
```

std_logic_1164 содержит выражение, которое разрешает использование пакета *std_logic_1164* стандартной библиотеки IEEE в составе формируемого модуля VHDL-описания. В указанном пакете определены базовые типы языка VHDL, операции над этими типами данных и функции преобразования типов:

```
use IEEE.std_logic_1164.all;
```

std_logic_misc представляет собой выражение, предоставляющее возможность использования в составе описания проектируемого устройства всех типов данных и функций одноименного пакета стандартной библиотеки IEEE. Пакет *std_logic_misc* содержит определения дополнительных типов, подтипов и соответствующих функций для пакета *std_logic_1164*:

```
use IEEE.std_logic_misc.all;
```

std_logic_signed является шаблоном, предназначенным для предоставления доступа к функциям пакета *std_logic_signed* в формируемом VHDL-описании. В указанном пакете определены операции и функции над данными, которые относятся к типу STD_LOGIC_VECTOR, в знаковом представлении:

```
use IEEE.std_logic_signed.all;
```

std_logic_textio включает в себя выражение, открывающее доступ к процедурам одноименного пакета стандартной библиотеки IEEE. Процедуры READ и WRITE, определенные в этом пакете, предназначены для организации файлового ввода/вывода данных при моделировании разрабатываемых устройств. Пакет *std_logic_textio* содержит также определения процедур HREAD и HWRITE, которые используются для организации файлового ввода/вывода данных, представленных в шестнадцатеричном формате:

```
use std.textio.all;
```

std_logic_unsigned содержит выражение, разрешающее использование функций пакета *std_logic_unsigned* в формируемом VHDL-описании. В этом пакете определены операции и функции над данными, которые относятся к типу STD_LOGIC_VECTOR, в беззнаковом представлении:

```
use IEEE.std_logic_unsigned.all;
```

User Defined представляет собой шаблон, предназначенный для создания ссылки на библиотеку и некоторый пакет этой библиотеки, название которых определяется разработчиком:

```
library <LIB_NAME>;
use <LIB_NAME>.<PACKAGE_NAME>.all
```

При использовании этого выражения <LIB_NAME> заменяется названием соответствующей библиотеки, а <PACKAGE_NAME> — наименованием требуемого пакета указанной библиотеки.

Шаблоны основных операторов языка VHDL

Шаблоны основных операторов языка VHDL содержатся в разделе *Operators*, который входит в состав папки *Common Constructs*. Подробная структура этого раздела приведена на рис. 8. Все шаблоны, представленные в разделе *Operators*, сгруппированы в соответствии с классификацией по типу выполняемых операций.

Arithmetic включает в себя список стандартных арифметических операторов языка VHDL. В состав данного списка входят операторы, которые предназначены для выполнения функций сложения, вычитания, умножения, деления, взятия по модулю и возведения в степень.

```
The following are the arithmetic operators as defined by the VHDL
language.
--
   + ---- Addition
   - ---- Subtraction
   * ---- Multiplication
   / ---- Divide
   mod ---- Modulus
   ** ---- Power Operator (i.e. 2**8 returns 256)
```

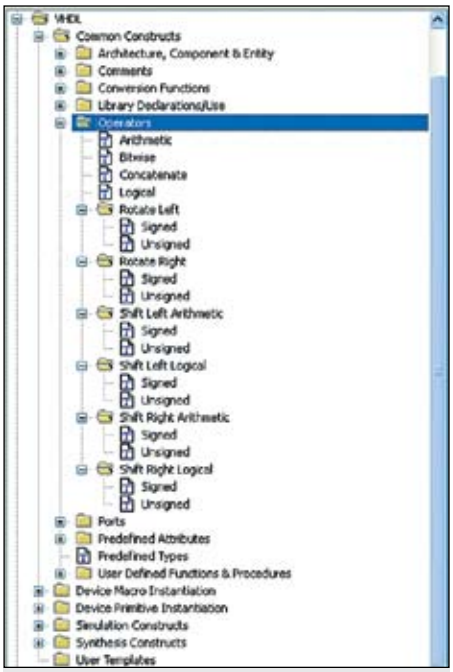

Рис. 8. Структура раздела Operators шаблонов базовых конструкций языка VHDL

Bitwise содержит группу основных поразрядных логических операторов языка VHDL. Данную группу образуют операторы, предназначенные для выполнения следующих функций:

- поразрядное «Логическое НЕ»;
- поразрядное «Логическое И»;
- поразрядное «Логическое ИЛИ»;
- поразрядное «Исключающее ИЛИ»;
- поразрядное «Исключающее ИЛИ-НЕ».

```
--The following operators can be used on two single bits to produce
a single bit
--output or two equivelent sized bused signals where the operations
are performed
--on each bit of the bus. In the case of the Invert, only one signal
or bus is
--provided and the operation occurs on each bit of the signal.
--
   NOT ---- Invert a single-bit signal or each bit in a bus
   AND ---- AND two single bits or each bit between two buses
   OR ---- OR two single bits or each bit between two buses
   XOR ---- XOR two single bits or each bit between two buses
   XNOR ---- XNOR two single bits or each bit between two buses
```

Concatenate представляет собой образец применения оператора конкатенации.

```
--The following operators either concatenates several bits into a bus
or replicate
--a bit or combination of bits multiple times.
--
   a & b & c ---- Concatenate a, b and c into a bus
```

В приведенном примере оператор конкатенации используется для объединения трех различных шин в одну.

Logical включает в себя группы стандартных логических операторов языка VHDL и операторов сравнения. Указанные операторы предназначены для использования в составе условных выражений. Группа логических операторов представлена операторами:

- «Логическое НЕ»;
- «Логическое И»;
- «Логическое ИЛИ».

В группу операторов сравнения входят:

- оператор равенства;
- оператор неравенства;
- оператор «меньше»;
- оператор «меньше или равно»;
- оператор «больше»;
- оператор «больше или равно».

```
--The following logical operators are used in conditional TRUE/FALSE
statements
--such as an if statement in order to specify the condition for the
operation.
--
   NOT ---- Not True
   AND ---- Both Inputs True
   OR ---- Either Input True
   = ---- Inputs Equal
   /= ---- Inputs Not Equal
   < ---- Less-than
   <= ---- Less-than or Equal
   > ---- Greater-than
   >= ---- Greater-than or Equal
```

Папка *Rotate Left*, входящая в состав раздела *Operators* (рис. 8), содержит шаблоны операторов циклического сдвига данных влево. Для применения всех операторов сдвига данных в состав формируемого VHDL-описания необходимо включить пакет *numeric_std* стандартной библиотеки IEEE.

Signed представляет собой шаблон использования оператора циклического сдвига данных влево для операндов со знаком.

```
--The IEEE.numeric_std library will need to be acessed for these
functions
--
<signed_sig> rol <shift_amount_in_integer>;
```

При включении данного шаблона в состав формируемого VHDL-описания нужно вместо <signed_sig> указать идентификатор переменной или сигнала знакового типа, над которыми выполняется операция циклического сдвига. Количество разрядов, на которое производится сдвиг, определяется с помощью параметра <shift_amount_in_integer>. В качестве этого параметра можно указывать имя константы, переменной или значение целого типа.

Unsigned содержит шаблон применения оператора циклического сдвига данных влево по отношению к сигналу или переменной беззнакового типа:

```
--The IEEE.numeric_std library will need to be acessed for these
functions
--
<unsigned_sig> rol <shift_amount_in_integer>;
```

При практическом использовании этого шаблона следует заменить <unsigned_sig> идентификатором переменной или сигнала беззнакового типа, над которым осуществляется операция циклического сдвига влево.

В папке *Rotate Right*, представленной в составе раздела *Operators*, находятся шаблоны операторов циклического сдвига данных вправо.

Signed является шаблоном варианта использования оператора циклического сдвига данных вправо для значений сигналов или переменных, относящихся к знаковому типу:

```
--The IEEE.numeric_std library will need to be acessed for these
functions
--
<signed_sig> ror <shift_amount_in_integer>;
```

Unsigned включает в себя шаблон применения оператора циклического сдвига данных вправо для значений сигналов или переменных, которые относятся к беззнаковому типу:

```
--The IEEE.numeric_std library will need to be acessed for these
functions
--
<unsigned_sig> ror <shift_amount_in_integer>;
```

Папка *Shift Left Arithmetic*, которая входит в состав раздела *Operators* (рис. 8), предоставляет шаблоны операторов арифметического сдвига данных влево.

Signed представляет собой шаблон варианта использования оператора арифметического сдвига данных влево для значений сигналов или переменных знакового типа:

```
--The IEEE.numeric_std library will need to be acessed for these functions
--
<signed_sig> sla <shift_amount_in_integer>;
```

Unsigned содержит шаблон варианта применения оператора арифметического сдвига данных влево для значений сигналов или переменных беззнакового типа:

```
--The IEEE.numeric_std library will need to be acessed for these functions
--
<unsigned_sig> sla <shift_amount_in_integer>;
```

В папке *Shift Left Logical*, представленной в разделе *Operators*, сосредоточены шаблоны операторов логического сдвига данных влево.

Signed включает в себя шаблон варианта использования оператора логического сдвига данных влево для значений сигналов или переменных со знаком:

```
--The IEEE.numeric_std library will need to be acessed for these functions
--
<signed_sig> sll <shift_amount_in_integer>;
```

Unsigned является шаблоном варианта применения оператора логического сдвига данных влево для значений сигналов или переменных без знака:

```
--The IEEE.numeric_std library will need to be acessed for these functions
--
<unsigned_sig> sll <shift_amount_in_integer>;
```

В папке *Shift Right Arithmetic*, входящей в состав раздела *Operators*, находятся шаблоны операторов арифметического сдвига данных вправо.

Signed содержит шаблон варианта использования оператора арифметического сдвига данных вправо для значений сигналов или переменных со знаком:

```
--The IEEE.numeric_std library will need to be acessed for these functions
--
<signed_sig> sra <shift_amount_in_integer>;
```

Unsigned представляет собой вариант применения оператора арифметического сдвига данных вправо для значений сигналов или переменных, которые принадлежат к беззнаковому типу:

```
--The IEEE.numeric_std library will need to be acessed for these functions
--
<unsigned_sig> sra <shift_amount_in_integer>;
```

В папке *Shift Right Logical* сгруппированы шаблоны операторов логического сдвига данных вправо.

Signed включает в себя шаблон варианта использования оператора логического сдвига данных вправо для значений сигналов или переменных, которые относятся к знаковому типу:

```
--The IEEE.numeric_std library will need to be acessed for these functions
--
<signed_sig> srl <shift_amount_in_integer>;
```

Unsigned является шаблоном варианта применения оператора логического сдвига данных вправо для значений сигналов или переменных беззнакового типа:

```
--The IEEE.numeric_std library will need to be acessed for these functions
--
<unsigned_sig> srl <shift_amount_in_integer>;
```

Шаблоны выражений декларации интерфейсных портов

Шаблоны выражений, предназначенных для декларации интерфейсных портов в формируемом VHDL-описании, сосредоточены в разделе *Ports*, развернутая структура которого показана на рис. 9.

Все шаблоны, представленные в этом разделе, сгруппированы в три папки в соответствии с направлением передачи данных через интерфейсные порты (режимом работы порта). Эти папки содержат шаблоны выражения объявления входных, выходных и двунаправленных портов с различной разрядностью, которые имеют тип *std_logic* или *std_logic_vector*. Этот тип интерфейсных портов наиболее часто применяется в VHDL-описаниях разрабатываемых устройств.

Папка *Bidirectional (inout)* включает в себя шаблоны выражений декларации двунаправленных портов с наиболее распространенной разрядностью.

1-it, 2-it, 3-it, 4-it, 8-it, 16-it, 32-it и ***64-it*** представляют собой шаблоны, предназначенные для объявления двунаправленных интерфейсных портов с соответствующей разрядностью:

```
<port_name>: inout std_logic;
<port_name>: inout std_logic_vector (1 downto 0);
<port_name>: inout std_logic_vector (2 downto 0);
<port_name>: inout std_logic_vector (3 downto 0);
<port_name>: inout std_logic_vector (7 downto 0);
<port_name>: inout std_logic_vector (15 downto 0);
<port_name>: inout std_logic_vector (31 downto 0);
<port_name>: inout std_logic_vector (63 downto 0);
```

В папке *Input*, входящей в состав раздела *Ports* (рис. 9), приведены шаблоны выражений, используемые для декларации входных портов.

1-it, 2-it, 3-it, 4-it, 8-it, 16-it, 32-it и ***64-it***, расположенные в указанной папке, являются шаблонами, предназначенными для объявления входных интерфейсных портов, разрядность которых отражена в названиях:

```
<port_name>: in std_logic;
<port_name>: in std_logic_vector (1 downto 0);
<port_name>: in std_logic_vector (2 downto 0);
<port_name>: in std_logic_vector (3 downto 0);
<port_name>: in std_logic_vector (7 downto 0);
<port_name>: in std_logic_vector (15 downto 0);
<port_name>: in std_logic_vector (31 downto 0);
<port_name>: in std_logic_vector (63 downto 0);
```

Папка *Output*, представленная в составе раздела *Ports*, содержит шаблоны выражений

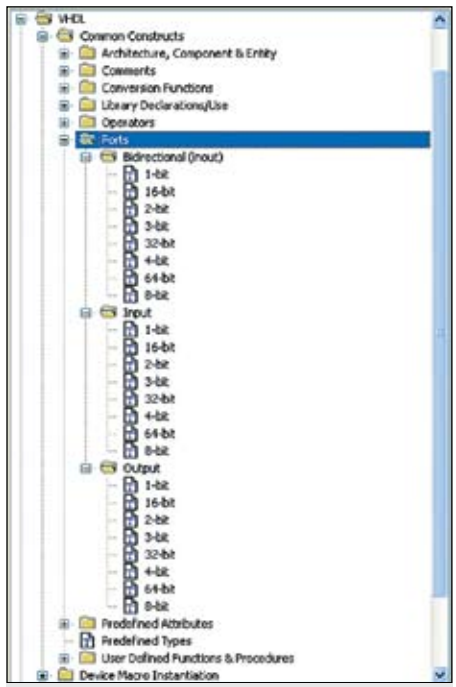

Рис. 9. Структура раздела Ports шаблонов базовых конструкций языка VHDL

декларации выходных интерфейсных портов с наиболее часто используемой разрядностью.

1-it, 2-it, 3-it, 4-it, 8-it, 16-it, 32-it и ***64-it***, находящиеся в данной папке, включают в себя шаблоны выражений, которые предназначены для объявления выходных портов с соответствующим количеством разрядов:

```
<port_name>: out std_logic;
<port_name>: out std_logic_vector (1 downto 0);
<port_name>: out std_logic_vector (2 downto 0);
<port_name>: out std_logic_vector (3 downto 0);
<port_name>: out std_logic_vector (7 downto 0);
<port_name>: out std_logic_vector (15 downto 0);
<port_name>: out std_logic_vector (31 downto 0);
<port_name>: out std_logic_vector (63 downto 0);
```

При практическом использовании шаблонов выражений декларации входных, выходных и двунаправленных портов, представленных выше, следует вместо <port_name> указать идентификатор соответствующего порта. Рассмотренные шаблоны можно использовать и в тех случаях, когда необходимо добавить в формируемое VHDL-описание выражения декларации интерфейсных портов с другой разрядностью. При этом следует изменить значения границ диапазона изменения индекса.

Шаблоны предопределенных атрибутов языка VHDL

Шаблоны предопределенных атрибутов языка VHDL находятся в разделе *Predefined Attributes*, развернутая структура которого изображена на рис. 10. Подробную информацию о назначении атрибутов массивов, объектов, сигналов и типов можно найти в [2, 3, 5, 14–16].

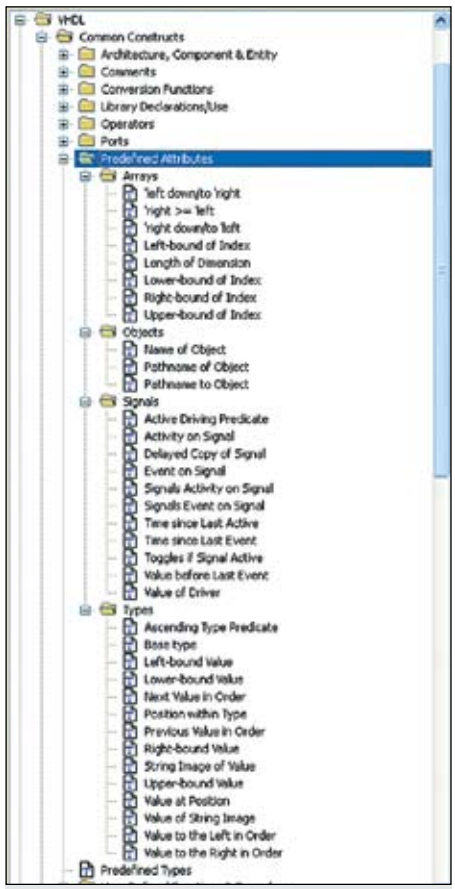

Рис. 10. Структура раздела Predefined Attributes базовых шаблонов языка VHDL

Все шаблоны, расположенные в этом разделе, распределены в четыре папки, названия которых определяют принадлежность атрибутов соответствующим элементам языка VHDL.

Папка *Arrays*, входящая в состав раздела *Predefined Attributes* шаблонов базовых конструкций языка, объединяет шаблоны атрибутов, которые относятся к массивам.

'left down/to 'right представляет собой шаблон записи атрибута *range*, значение которого определяет диапазон изменения индекса массива указанной размерности:

```
<array_id>'range (<expr>)
```

После включения этого шаблона в состав формируемого VHDL-описания следует вместо <array_id> задать имя массива, к которому относится данный атрибут, а вместо <expr> — соответствующую размерность этого массива.

'right >= 'left содержит шаблон выражения, в котором используется атрибут *ascending*, значение которого определяет направление изменения индекса массива заданной размерности (возрастающее или убывающее):

```
<array_id>'ascending (<expr>)
```

'right down/to 'left включает в себя шаблон записи атрибута *reverse_range*, значением которого является перевернутый по направлению и границам диапазон изменения индекса массива указанной размерности:

```
<array_id>'reverse_range (<expr>)
```

Left-bound of Index является шаблоном выражения, демонстрирующего применение атрибута *left*, значение которого соответствует левой границе диапазона изменения индекса массива заданной размерности:

```
<array_id>'left (<expr>)
```

Length of Dimension представляет собой шаблон записи атрибута *length*, который возвращает значение длины диапазона изменения индекса массива указанной размерности:

```
<array_id>'length (<expr>)
```

Lower-bound of Index содержит шаблон выражения, в котором используется атрибут *low*, значение которого информирует о нижней границе диапазона изменения индекса массива заданной размерности:

```
<array_id>'low (<expr>)
```

Right-bound of Index включает в себя шаблон записи атрибута *right*, значение которого соответствует правой границе диапазона изменения индекса массива заданной размерности:

```
<array_id>'right (<expr>)
```

Upper-bound of Index является шаблоном выражения, использующего атрибут *high*, который возвращает значение, соответствующее левой границе диапазона изменения индекса массива заданной размерности:

```
<array_id>'high (<expr>)
```

В папке *Objects*, которая входит в состав раздела *Predefined Attributes* базовых шаблонов языка, сгруппированы шаблоны атрибутов, которые характеризуют объекты VHDL-описаний.

Name of Object представляет собой шаблон записи атрибута *simple_name*, информирующего об идентификаторе соответствующего объекта:

```
<object_id>'simple_ name
```

Pathname of Object содержит шаблон выражения, в котором используется атрибут *instance_name*, возвращающий строку полного имени указанного объекта с учетом иерархии:

```
<object_id>'instance_name
```

Pathname to Object включает в себя шаблон записи атрибута *path_name*, который предоставляет информацию об иерархической части полного имени соответствующего объекта:

```
<object_id>'path_name
```

После включения требуемого шаблона атрибутов, характеризующих объекты VHDL-описаний, в состав создаваемого модуля описания нужно вместо <object_id> указать идентификатор соответствующего объекта.

В папке *Signals*, представленной в составе раздела *Predefined Attributes* базовых шаблонов языка VHDL, сосредоточены шаблоны атрибутов сигналов. При практическом использовании данных шаблонов следует заменить в них <signal_id> идентификатором соответствующего сигнала.

Active Driving Predicate включает в себя шаблон выражения использования атрибута *driving*, значение которого информирует о наличии или отсутствии изменения задающего значения сигнала в текущем цикле моделирования:

```
<signal_id>'driving
```

Activity on Signal предоставляет шаблон записи атрибута *active*, значение которого информирует о том, есть ли обращение к указанному сигналу в текущем цикле моделирования:

```
<signal_id>'active
```

Delayed Copy of Signal содержит шаблон выражения использования атрибута *delayed*, значение которого представляет собой копию указанного сигнала, задержанную на заданное время:

```
<signal_id>'delayed (<TIME>)
```

При включении этого шаблона в состав формируемого VHDL-описания нужно вместо <TIME> вставить требуемое значение задержки.

Event on Signal является шаблоном записи атрибута *event*, значение которого информирует об изменении уровня указанного сигнала в текущем цикле:

```
<signal_id>'event
```

Signals Activity on Signal представляет собой шаблон использования атрибута *quiet*, значение которого отражает наличие или отсутствие обращений к соответствующему сигналу в течение указанного времени:

```
<signal_id>'quiet (<TIME>)
```

В этом шаблоне параметр <TIME> задает значение временного интервала, в котором производится анализ обращений к сигналу.

Signals Event on Signal включает в себя шаблон записи атрибута *stable*, значение которого информирует о стабильности или нестабильности значения указанного сигнала в течение заданного интервала времени:

```
<signal_id>'stable (<TIME>)
```

Time since Last Active содержит шаблон выражения использования атрибута *last_active*, который возвращает значение интервала времени, прошедшего с момента последнего обращения к указанному сигналу:

```
<signal_id>'last_active
```

Time since Last Event является шаблоном записи атрибута *last_event*, который принимает значение интервала времени, прошедшего с момента последнего переключения уровня указанного сигнала:

```
<signal_id>'last_event
```

Toggles if Signal Active представляет собой шаблон выражения использования атрибута *transaction*, значение (битового типа) которого переключается в тех циклах, где происходит изменение уровня заданного сигнала:

```
<signal_id>'transaction
```

Value before Last Event включает в себя шаблон записи атрибута *last_value*, значение которого информирует о предыдущем уровне указанного сигнала, который присутствовал непосредственно перед последним переключением этого сигнала:

```
<signal_id>'last_value
```

Value of Driver содержит шаблон выражения использования атрибута *driving_value*, который возвращает задающее значение соответствующего сигнала:

```
<signal_id>'driving_value
```

Папка *Types*, входящая в состав раздела *Predefined Attributes* шаблонов базовых конструкций языка VHDL, объединяет в себе шаблоны атрибутов типов данных. После включения шаблонов этой группы в состав формируемого VHDL-описания нужно вместо <type_id> в этих выражениях указать идентификатор соответствующего типа данных.

Ascending Type Predicate представляет собой шаблон записи атрибута *ascending*, который возвращает значение типа BOOLEAN, соответствующее направлению изменения диапазона данных указанного типа или перечисляемому способу определения этого типа:

```
<type_id>'ascending
```

Base Type является шаблоном выражения использования атрибута *base*, значение которого соответствует базовому типу для указанного подтипа данных:

```
<type_id>'base
```

Left-bound Value содержит шаблон записи атрибута *left*, который возвращает значение левой границы диапазона (первое значение) соответствующего типа данных:

```
<type_id>'left
```

Lower-bound Value представляет собой шаблон выражения использования атрибута *low*, который позволяет получить информацию о наименьшем значении в указанном типе данных:

```
<type_id>'low
```

Next Value in Order включает в себя шаблон записи атрибута *succ*, который принимает значение соответствующего типа, расположенное в позиции, большей на единицу по сравнению с указанной в виде выражения <expr>:

```
<type_id>'succ (<expr>)
```

Position within Type является шаблоном выражения использования атрибута *pos*, который возвращает значение номера позиции заданного значения в указанном типе данных. После включения этого шаблона в состав создаваемого VHDL-описания нужно заменить <expr> соответствующим значением, для которого необходимо определить номер позиции:

```
<type_id>'pos (<expr>)
```

Previous Value in Order содержит шаблон записи атрибута *pred*, который принимает значение указанного типа, соответствующее позиции, меньшей на единицу по сравнению с заданной в форме выражения <expr>:

```
<type_id>'pred (<expr>)
```

Right-bound Value предоставляет шаблон выражения использования атрибута *right*, который возвращает значение правой границы диапазона (последнее значение) соответствующего типа данных:

```
<type_id>'right
```

String Image of Value включает в себя шаблон записи атрибута *image*, предназначенного для формирования строкового представления заданного значения <expr>, принадлежащего к указанному типу:

```
<type_id>'image (<expr>)
```

Upper-bound Value является шаблоном применения атрибута *high*, который позволяет получить информацию о наибольшем значении в соответствующем типе данных:

```
<type_id>'high
```

Value at Position представляет собой шаблон записи атрибута *val*, который возвращает значение указанного типа, соответствующее заданной позиции <expr>:

```
<type_id>'val (<expr>)
```

Value of String Image содержит шаблон выражения использования атрибута *value*, который предназначен для формирования значения указанного типа из заданного строкового представления <string>:

```
<type_id>'value (<string>)
```

Value to Left in Order включает в себя шаблон записи атрибута *leftof*, который возвращает значение соответствующего типа, расположенное слева от заданной позиции <expr>:

```
<type_id>'leftof (<expr>)
```

Value to Right in Order является шаблоном выражения использования атрибута *rightof*, который предоставляет значение указанного типа, находящееся справа от заданной позиции <expr>:

```
<type_id>'rightof (<expr>)
```

В папке *Common Constructs* представлен также модуль *Predefined Types* (рис. 10), в котором перечислены предопределенные типы данных языка VHDL. Для каждого из этих типов в виде комментариев приведена информация о значениях, которые могут принимать данные соответствующего типа:

```
STD_LOGIC              --'U','X','0','1','Z','W','L','H','-'
STD_LOGIC_VECTOR       --Natural Range of STD_LOGIC
BOOLEAN                --True or False
INTEGER                --32 or 64 bits
NATURAL                --Integers >= 0
POSITIVE               --Integers > 0
REAL                   --Floating-point
BIT                    --'0','1'
BIT_VECTOR(Natural)    --Array of bits
CHARACTER              --7-bit ASCII
STRING(POSITIVE)       --Array of charachters
TIME                   --hr, min, sec, ms, us, ns, ps, fs
DELAY_LENGTH           --Time >= 0
```

Шаблоны конструкций, предназначенных для декларации и обращения к функциям и процедурам, определяемых разработчиком

Шаблоны конструкций, которые используются для декларации и обращения к функциям и процедурам, определяемым разработчиком, расположены в разделе *User Defined Functions & Procedures*. Подробная структура этого раздела показана на рис. 11.

Calling a Function (named notation) содержит шаблон выражения вызова функции, определяемой пользователем, в котором используется ключевое соответствие формальных параметров и их фактических значений:

```
<signal_name> = <function_name> (<input1> => <insig1>,
<input2> => <insig2>, ...);
```

При практическом применении этого шаблона нужно вместо <function_name> указать название требуемой функции, а вместо <signal_name> — идентификатор сигнала или переменной, которым должно присваиваться значение вызываемой функции.

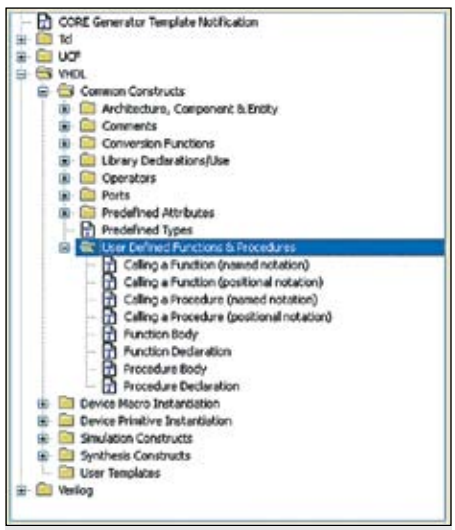

Рис. 11. Структура раздела User Defined Functions & Procedures базовых шаблонов языка VHDL

В списке параметров данной функции нужно заменить <input1>, <input2>, …, <inputN> идентификаторами формальных параметров, а <insig1>, <insig2>, …, <insigN> — их фактическими значениями.

Calling a Function (positional notation) представляет собой шаблон выражения обращения к функции, определяемой пользователем, в котором применяется позиционное соответствие формальных параметров и их фактических значений:

```
<signal_name> = <function_name> (<comma_separated_inputs>);
```

После включения данного шаблона в состав формируемого VHDL-описания необходимо вместо <comma_separated_inputs> вставить последовательность фактических значений параметров, разделенных запятыми, в том же порядке, как они перечислены в выражении декларации вызываемой функции.

Calling a Procedure (named notation) является шаблоном выражения вызова процедуры, определяемой пользователем, в котором используется ключевое соответствие формальных параметров и их фактических значений.

```
<procedure_name> (<input1> => <insig1>, ..., <output1> =>
<outsig1>, ...);
```

В данном шаблоне следует заменить <procedure_name> названием процедуры, к которой производится обращение. В списке параметров вызываемой процедуры нужно вместо <input1>, …, <inputN> и <output1>, …,<outputN> указать идентификаторы формальных входных и выходных параметров, а вместо <insig1>, …, <insigN> и <outsig1>, …, < outsigN> — фактические значения соответствующих параметров.

Calling a Procedure (positional notation) включает в себя шаблон выражения обращения к процедуре, определяемой разработчиком, в котором применяется позиционное соответствие формальных параметров и их фактических значений:

```
<procedure_name> (<comma_separated_inputs>,<comma_
separated_outputs>);
```

После включения данного шаблона в состав создаваемого VHDL-описания следует заменить <comma_separated_inputs> списком фактических значений входных параметров, а <comma_separated_outputs> — списком идентификаторов фактических значений выходных параметров. При этом должен соблюдаться тот же порядок перечисления фактических значений параметров, что и в выражении декларации вызываемой процедуры.

Function Body содержит конструкцию, предназначенную для определения пользовательской функции:

```
function <FUNC_NAME> ( <comma_separated_inputs> : <type>;
                       <comma_separated_inputs> : <type>)
                                                  return <type> is
  -- subprogram_declarative_items (constant declarations, variable
declarations, etc.)
  begin
    -- function body
  end <FUNC_NAME>;
```

Для формирования законченного описания функции на базе этого шаблона необходимо указать название определяемой функции вместо <FUNC_NAME>, тип возвращаемого значения <type> после ключевого слова return, список идентификаторов входных параметров <comma_separated_inputs> и их тип <type>. После строки комментария *subprogram_declarative_items* следует поместить выражения декларации локальных переменных и констант. Совокупность выражений, образующих тело функции, нужно добавить после строки комментария *function body*.

Function Declaration представляет собой шаблон конструкции, которая предназначена для декларации функции, определяемой пользователем. После включения этого шаблона в состав формируемого описания следует заменить <FUNC_NAME> названием объявляемой функции, а <comma_separated_inputs> — идентификатором соответствующего входного формального параметра, тип которого указывается вместо <type>. В заключительной части шаблона необходимо заменить <type> обозначением типа возвращаемого значения декларируемой функции:

```
function <FUNC_NAME> (<comma_separated_inputs> : <type>;
                      <comma_separated_inputs> : <type>)
                                                  return <type>;
```

Procedure Body включает в себя конструкцию, предназначенную для определения пользовательской процедуры:

```
rocedure <PROC_NAME> ( <comma_separated_inputs> : in <type>;
                       <comma_separated_outputs> : out <type>) is
    -- subprogram_declarative_items (constant declarations, variable
declarations, etc.)
  begin
    -- procedure body
  end <PROC_NAME>;
```

При использовании этого шаблона необходимо указать название процедуры вместо <PROC_NAME>, список входных параметров вместо <comma_separated_inputs> и их тип вместо <type>, список выходных параметров вместо <comma_separated_outputs> и их тип. Затем после строки комментария *subprogram_declarative_items* нужно добавить, при необходимости, выражения декларации локальных переменных и констант. Последовательность выражений, составляющих содержание процедуры, следует вставить между строкой комментария *procedure body* и ключевым словом *end*.

Procedure Declaration является конструкцией, применяемой для осуществления декларации процедуры, определяемой разработчиком:

```
procedure <PROC_NAME> (<comma_separated_inputs> : in <type>;
                       <comma_separated_outputs> : out
                                                   <type>);
```

При использовании этого шаблона необходимо вставить название объявляемой процедуры вместо <PROC_NAME>, а также перечислить идентификаторы входных и выходных параметров вместо <comma_separated_inputs> и <comma_separated_outputs> соответственно, указав их тип вместо <type>.

Продолжение следует

Литература

1. Кузелин М. О., Кнышев Д. А., Зотов В. Ю. Современные семейства ПЛИС фирмы Xilinx. Справочное пособие. М.: Горячая линия – Телеком, 2004.
2. Бибило П. Н. Основы языка VHDL. М.: Солон-Р, 2000.
3. Бибило П. Н. Синтез логических схем с использованием языка VHDL. М.: Солон-Р, 2002.
4. Уэйкерли Дж. Ф.. Проектирование цифровых устройств. Том 1. М.: Пост-маркет, 2002.
5. Поляков А. К. Языки VHDL и Verilog в проектировании цифровой аппаратуры. М.: Солон-Пресс, 2003.
6. Зотов В. Инструментальный комплект Spartan-3 Starter Kit для практического освоения методов проектирования встраиваемых микропроцессорных систем на основе ПЛИС семейств FPGA фирмы Xilinx // Компоненты и технологии. 2005. № 7.
7. Зотов В. Новый инструментальный комплект Spartan-3E Starter Kit для практического освоения методов проектирования встраиваемых микропроцессорных систем на основе ПЛИС семейств FPGA фирмы Xilinx // Компоненты и технологии. 2006. № 10.
8. Зотов В. Новый инструментальный комплект Spartan-3A Starter Kit для практического освоения методов проектирования и отладки цифровых устройств с аппаратной и программной реализацией операций, реализуемых на основе ПЛИС семейств FPGA фирмы Xilinx // Компоненты и технологии. 2007. № 9.
9. Зотов В. Новый инструментальный комплект от компании Avnet на основе ПЛИС FPGA семейства Spartan-3A фирмы Xilinx // Компоненты и технологии. 2008. № 8.
10. Зотов В. Инструментальный модуль компании Avnet для отладки проектов встраиваемых систем, разрабатываемых на базе нового семейства ПЛИС FPGA фирмы Xilinx Virtex-5 FXT // Компоненты и технологии. 2008. № 9.
11. Зотов В. Особенности архитектуры нового поколения высокопроизводительных ПЛИС FPGA фирмы Xilinx серии Virtex-6 // Компоненты и технологии. 2009. № 8.
12. Зотов В. Особенности архитектуры нового поколения ПЛИС FPGA фирмы Xilinx серии Spartan-6 // Компоненты и технологии. 2009. № 9.
13. Зотов В. Новое семейство высокопроизводительных ПЛИС с архитектурой FPGA фирмы Xilinx Virtex-6 HXT // Компоненты и технологии. 2010. № 1.
14. IEEE Standard VHDL Language Reference Manual — IEEE Std 1076-2002.
15. Суворова Е. А., Шейнин Ю. Е. Проектирование цифровых систем на VHDL. СПб.: БХВ-Петербург, 2003.
16. Сергиенко А. М. VHDL для проектирования вычислительных устройств. Киев: ООО «ТИД “ДС”», 2003.