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

Валерий ЗОТОВ
walerry@km.ru

Языки описания аппаратуры HDL (Hardware Description Language) в настоящее время являются основным средством представления цифровых устройств при их проектировании. В значительной степени это обусловлено широким применением программируемых логических интегральных схем (ПЛИС) в качестве элементной базы для создания цифровых систем, а также постоянным совершенствованием соответствующих инструментов синтеза и средств HDL-моделирования. Кроме того, при реализации больших проектов на базе современных ПЛИС проявляются ограничения схемотехнического метода описания цифровых устройств, которые снимаются при использовании языков высокого уровня VHDL и Verilog. Эти ограничения наиболее заметны при использовании кристаллов программируемой логики с архитектурой FPGA (Field Programmable Gate Array) [1].

Применение языков описания аппаратуры высокого уровня также повышает мобильность разрабатываемых проектов, так как большинство систем проектирования различных производителей поддерживает стандарты VHDL и Verilog. Поэтому проекты, представленные в виде HDL-описаний, с минимальными изменениями переносятся из одной системы проектирования в другую. Необходимость такого переноса часто возникает при смене элементной базы для реализации проектируемого устройства.

На сегодняшний день имеется ряд изданий [2–5], посвященных основам языка VHDL и примерам его использования, которые предоставляют разработчикам возможность самостоятельного освоения этого средства описания разрабатываемых цифровых устройств. После изучения теоретических основ данного языка при переходе к практике создания описаний проектов использование инструмента шаблонов позволяет упростить и ускорить процесс обучения. В состав пакетов современных САПР, в частности ISE (Integrated Software Environment/Integrated Synthesis Environment) Design Suite фирмы Xilinx, входит редактор HDL-кода, который, помимо простейших операций редактирования (копирования и вставки текста), поддерживает механизм шаблонов. Эти шаблоны могут эффективно использоваться не только в процессе изучения языка, но и в последующей практической работе.

Применение шаблонов, поддерживаемых встроенным HDL-редактором САПР серии Xilinx ISE Design Suite, предоставляет разработчикам целый ряд преимуществ, среди которых наиболее существенными являются:

- сокращение времени создания исходных описаний проектируемого устройства;
- исключение синтаксических ошибок, появление которых возможно при наборе текста описания вручную, с помощью клавиатуры;
- возможность использования готовых отлаженных конструкций для представления наиболее часто встречающихся элементов и функциональных блоков разрабатываемых устройств;
- поддержка формирования описаний специализированных аппаратных модулей, входящих в состав архитектуры кристаллов программируемой логики перспективных серий Virtex и Spartan [6–13];
- возможность создания и использования в дальнейшей работе собственных отработанных конструкций.

Прежде чем перейти непосредственно к изучению шаблонов языка VHDL, рассмотрим процедуру создания исходного HDL-описания разрабатываемого устройства в САПР Xilinx ISE Design Suite.

Создание описания проектируемого устройства на языке VHDL в САПР серии Xilinx ISE Design Suite

Создание VHDL-описания разрабатываемого устройства или его функциональных блоков начинается с выполнения процедуры подготовки основы нового исходного модуля проекта, которая активизируется кнопкой на панели инструментов, расположенной слева от окна исходных модулей ***Sources***, или командой ***New Source*** из раздела ***Project*** основного меню навигатора проекта (***Project Navigator***). При этом производится запуск мастера формирования основы нового исходного модуля описания ***New Source Wizard***. Работа мастера начинается с открытия стартовой диалоговой панели ***Select Source Type***, показанной на рис. 1. Данная панель позволяет выбрать тип нового модуля, задать его имя и указать место расположения создаваемого файла на диске.

В диалоговой панели ***Select Source Type*** прежде всего нужно указать тип создаваемого исходного модуля (соответствующая строчка в списке в левой части панели). Чтобы сформировать файл исходного VHDL-описания проектируемого устройства, в качестве типа нового модуля необходимо выбрать строку ***VHDL Module***. Затем активизировать поле редактирования названия модуля ***File Name***

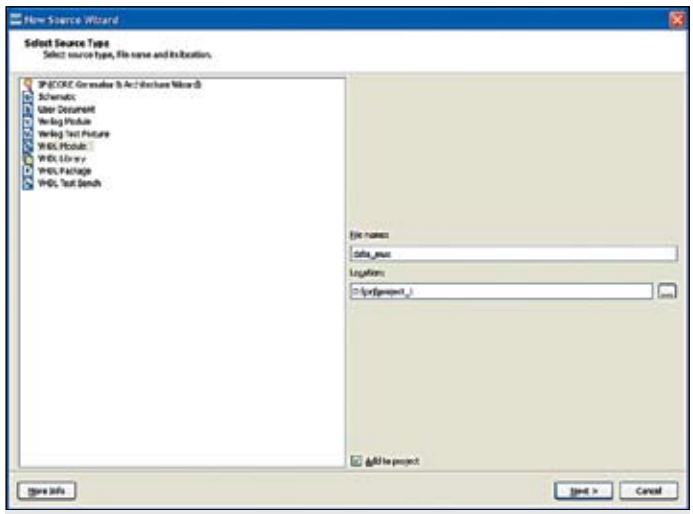

Рис. 1. Выбор типа создаваемого файла в стартовой диалоговой панели Select Source Type «мастера» формирования основы нового исходного модуля описания New Source Wizard

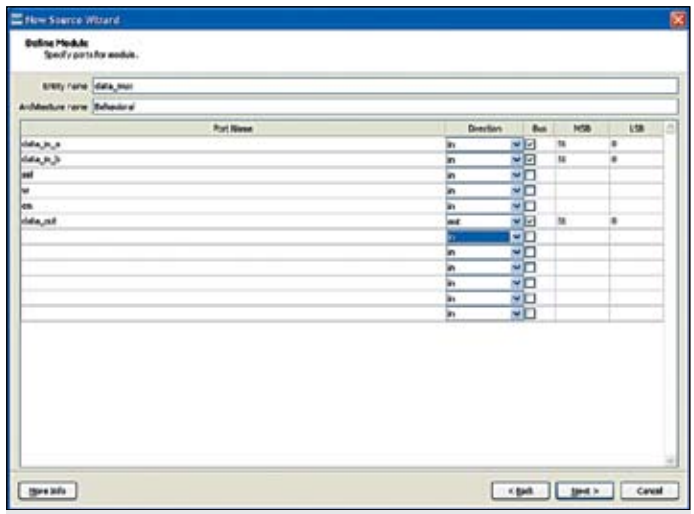

Рис. 2. Определение исходных данных для создаваемого VHDL-описания с помощью диалоговой панели Define Module «мастера» формирования основы нового исходного модуля New Source Wizard

и ввести имя создаваемого файла. В составе идентификатора формируемого модуля VHDL-описания могут использоваться только заглавные и строчные буквы латинского алфавита (A–Z, a–z), цифры (0–9) и символ подчеркивания (_). Место расположения создаваемого модуля на диске указывается в поле редактирования ***Location***. По умолчанию в этом поле редактирования предлагается название рабочего каталога текущего проекта, включающее полный путь доступа к нему. Для выбора иного каталога следует воспользоваться клавиатурой или диалоговой панелью навигации, которая выводится на экран при нажатии кнопки с пиктограммой «…», расположенной справа от поля редактирования ***Location***.

После ввода названия создаваемого файла VHDL-описания и нажатия кнопки «Далее» (***Next***), расположенной в нижней части диалоговой панели ***Select Source Type***, открывается следующая диалоговая панель мастера формирования основы нового исходного модуля описания ***New Source Wizard***, именуемая ***Define Module***. Вид этой диалоговой панели представлен на рис. 2.

Диалоговая панель ***Define Module*** предоставляет возможность определения в наглядной форме исходных данных для создаваемого VHDL-описания. Содержит два поля редактирования — ***Entity Name*** и ***Architecture Name***, а также таблицу, в которой должна быть представлена вся необходимая информация об интерфейсных портах описываемого объекта.

В поле редактирования ***Entity Name*** необходимо указать имя объекта. По умолчанию предлагается идентификатор, совпадающий с названием создаваемого модуля. Имя архитектурного тела VHDL-описания указывается в поле редактирования ***Architecture Name***. По умолчанию в качестве имени архитектурного тела предлагается идентификатор ***Behavioral***. Он предполагает, что в описании будет использован поведенческий стиль. При необходимости с помощью клавиатуры можно изменить данный идентификатор.

Далее следует заполнить таблицу описания портов, которая содержит пять столбцов. В ячейки первого столбца, озаглавленного ***Port Name***, заносится идентификатор порта. В колонке ***Direction*** указывается тип порта для передачи данных. Каждая ячейка данного столбца представляет собой поле выбора, выпадающий список которого содержит три возможных варианта, определяющих тип порта: ***in*** (входной), ***out*** (выходной) или ***inout*** (двунаправленный). В третьей колонке (***Bus***) указывается информация о структуре сигнала, который ассоциируется с описываемым портом. Если этот сигнал имеет шинную структуру, то следует переключить индикатор в состояние «Включено». Колонки ***MSB*** и ***LSB*** заполняются только для портов, представленных в виде шин и описываемых с помощью векторов. В столбце ***MSB*** указываются значения индекса, соответствующего старшему разряду шины (вектора), а в ***LSB*** — младшему. Если описание портов нового модуля не помещается в видимой части таблицы, следует воспользоваться элементами вертикальной прокрутки.

После внесения всех необходимых данных следует нажать кнопку «Далее» (***Next***) в нижней части диалоговой панели ***Define Module***, в результате чего открывается заключительная информационная панель ***Summary*** «мастера» формирования основы нового исходного модуля описания ***New Source Wizard***. В этой панели, вид которой приведен на рис. 3, отображается вся информация, на базе которой выполняется автоматическое формирование основы нового модуля VHDL-описания.

Если все данные, необходимые для создания основы нового VHDL-описания, указаны корректно, нужно нажать кнопку «Готово» (***Finish***), в результате чего в навигаторе проекта (***Project Navigator***) открывается новое рабочее окно встроенного HDL-редактора (рис. 4), в котором отображается автоматически сформированный код.

Сгенерированный код основы нового VHDL-описания проектируемого устройства включает декларацию используемых библиотек и пакетов, интерфейса описываемого объекта ***entity*** и основу архитектурного

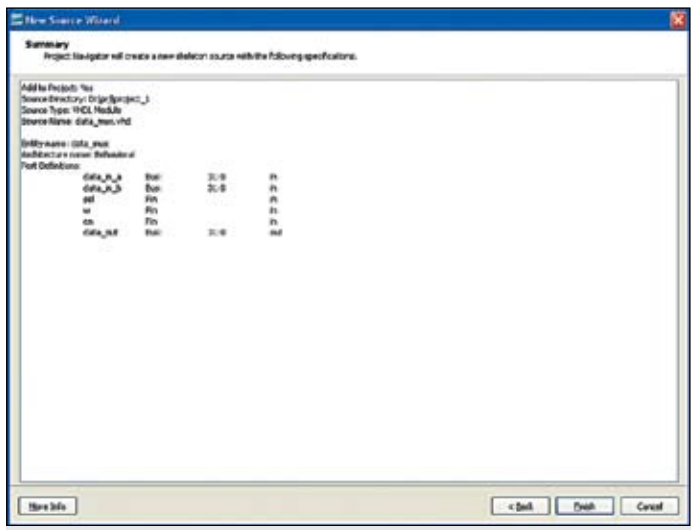

Рис. 3. Заключительная информационная панель Summary «мастера» формирования основы нового исходного модуля описания New Source Wizard

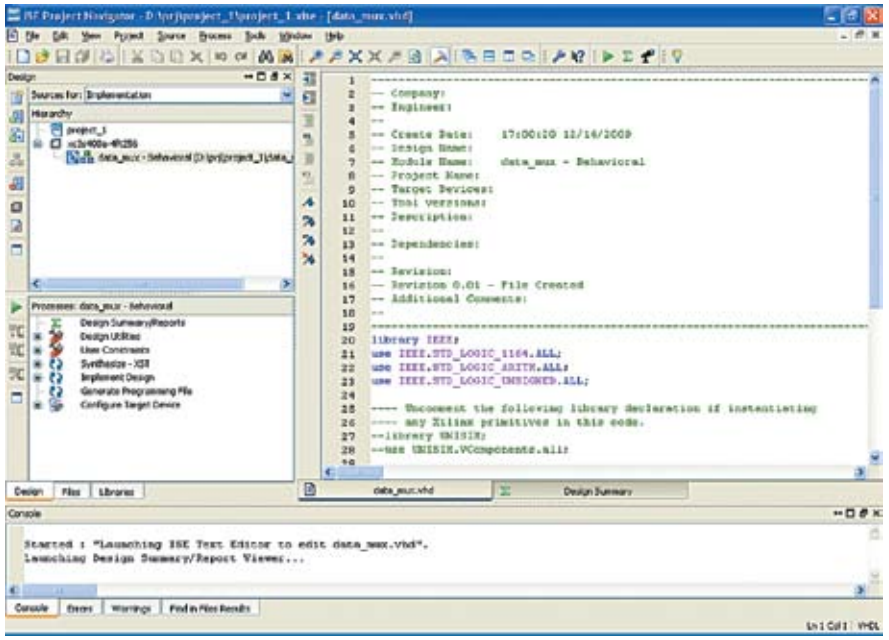

Рис. 4. Навигатор проекта после успешного формирования основы нового VHDL-описания проектируемого устройства

тела VHDL-описания. В качестве примера ниже приведен автоматически сформированный текст основы описания устройства ***data_mux***, который имеет две 32-разрядные входные шины данных ***data_in_a*** и ***data_in_b***, входы управления ***sel***, ***w***, ***en***, а также 32-разрядную выходную шину данных ***data_out***.

```
----------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:         17:00:20 12/14/2009
-- Design Name:
-- Module Name:         data_mux - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
----
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
--
entity data_mux is
        Port (
                data_in_a : in STD_LOGIC_VECTOR (31 downto 0);
                data_in_b : in STD_LOGIC_VECTOR (31 downto 0);
                sel : in STD_LOGIC;
                w : in STD_LOGIC;
                en : in STD_LOGIC;
                data_out : out STD_LOGIC_VECTOR (31 downto 0)
              );
end data_mux;
--
architecture Behavioral of data_mux is
--
begin
----
end Behavioral;
```

В начале автоматически сформированного текста представлено несколько строк комментариев, в которых можно указать справочную информацию о создаваемом исходном модуле, в том числе сведения о разработчике, времени создания и версии описания. Далее следуют ссылки на используемую библиотеку IEEE и ее стандартные логические пакеты STD_LOGIC_1164,IEEE. STD_LOGIC_ARITH, IEEE. STD_LOGIC_UNSIGNED. Кроме того, в виде комментариев приведены ссылки на библиотеку UNISIM, предоставляемую фирмой Xilinx, и ее пакет VComponents. Данные ссылки нужно активизировать (убрать символы комментариев) в том случае, если в составе создаваемого описания проектируемого устройства будут использоваться библиотечные компоненты, которые определены в указанном пакете.

После ссылок на применяемые библиотеки помещено объявление объекта описания с указанным ранее именем и описание его интерфейса на основе данных о портах устройства, перечисленных в табличной форме в диалоговой панели ***Define Module*** «мастера» формирования основы нового исходного модуля описания ***New Source Wizard*** (рис. 2). Завершает текст автоматически сгенерированной основы модуля VHDL-описания шаблон блока определения архитектуры описываемого объекта. В этом блоке перед ключевым словом ***begin*** нужно поместить выражения декларации всех используемых внутренних сигналов, констант, переменных и компонентов, а также определения функций и процедур, которые будут вызываться при описании архитектуры объекта. Для получения законченного описания на языке VHDL необходимо после ключевого слова ***begin*** до заключительной строки ***end Behavioral*** с помощью параллельно выполняемых операторов языка VHDL добавить определение функционирования или структуры объекта, который представляет разрабатываемое устройство. Текст описания архитектуры проектируемого устройства вводится с помощью клавиатуры. При этом рекомендуется использовать шаблоны встроенного HDL-редактора пакета САПР серии Xilinx ISE Design Suite.

Модуль VHDL-описания проекта следует сохранить в виде файла, используя для этого команды ***Save*** или ***Save As*** из всплывающего меню ***File*** или кнопку, расположенную на оперативной панели управления ***Project Navigator***.

Создание VHDL-пакета в САПР серии Xilinx ISE

Для оптимального многократного использования собственных типов данных, констант, функций и процедур в различных проектах целесообразно поместить их декларации и определения в отдельный пакет. Чтобы применять все эти элементы в новом проекте, достаточно включить в его состав соответствующий сформированный пакет. В общем случае в структуре описания VHDL-пакета можно выделить три части:

- блок ссылок на используемые библиотеки и пакеты;
- блок декларации элементов пакета;
- тело пакета.

Первая часть необязательна. Она применяется в случаях, когда в описании VHDL-пакета используются элементы, которые определяются в других библиотеках и пакетах. Во второй части, которая начинается с ключевого слова ***package*** и завершается ключевым словом ***end*** с указанием названия пакета, производится декларация всех элементов создаваемого пакета, которые должны быть доступны пользователям этого пакета. Третья часть начинается с сочетания ключевых слов ***package body*** и заканчивается ключевым словом ***end*** с указанием названия этого пакета. В теле пакета приводятся определения функций и процедур, которые были объявлены в предыдущей части формируемого VHDL-пакета. Здесь же могут присутствовать выражения декларации локальных типов данных и констант, которые используются только в теле пакета и недоступны за его пределами.

Создание пакета начинается так же, как и подготовка основы нового исходного модуля VHDL-описания, с нажатия кнопки на панели инструментов или активизации команды ***New Source*** из раздела ***Project*** основного меню ***Project Navigator***. Далее в панели ***Select Source Type*** нужно выбрать строку ***VHDL Package*** и указать в поле редактирования ***File Name*** название создаваемого пакета. После нажатия кнопок «Далее» (***Next***) и «Готово» (***Finish***) в появившейся вслед

за этим панели ***Summary*** автоматически открывается новое рабочее окно интегрированного HDL-редактора, в котором отображается автоматически сформированный шаблон описания VHDL-пакета. Текст шаблона выглядит следующим образом:

```
-- Package File Template
--
-- Purpose: This package defines supplemental types, subtypes,
constants, and functions
--
library IEEE;
use IEEE.STD_LOGIC_1164.all;
--
package <Package_Name> is
--
type <new_type> is
   record
      <type_name>: std_logic_vector( 7 downto 0); <type_name> :
std_logic;
   end record;
--
-- Declare constants
--
constant <constant_name>   : time := <time_unit> ns;
constant <constant_name>   : integer := <value>;
--
-- Declare functions and procedure
--
function <function_name> (signal <signal_name> : in <type_
declaration>) return <type_declaration>;
procedure <procedure_name> (<type_declaration> <constant_name>
: in <type_declaration>);
--
end <Package_Name>;
--
--
package body <Package_Name> is
--
-- Example 1
function <function_name> (signal <signal_name> : in <type_
declaration> ) return <type_declaration> is
   variable <variable_name> : <type_declaration>;
begin
   <variable_name> := <signal_name> xor <signal_name>; return
<variable_name>;
end <function_name>;
--
--
-- Example 2
function <function_name> (signal <signal_name> : in <type_
declaration>; signal <signal_name> : in <type_declaration> ) return
<type_declaration> is
begin
   if (<signal_name> = '1') then
      return <signal_name>;
   else
      return 'Z';
   end if;
end <function_name>;
--
-- Procedure Example
procedure <procedure_name> (<type_declaration> <constant_name>
: in <type_declaration>) is
--
begin
--
end <procedure_name>;
--
end <Package_Name>;
```

Автоматически сгенерированный шаблон описания VHDL-пакета начинается со ссылок на используемую библиотеку IEEE и стандартный логический пакет STD_LOGIC_1164 этой библиотеки. При необходимости эту часть описания пакета можно дополнить другими ссылками на нужные библиотеки и пакеты. Далее после ключевого слова ***package*** нужно вместо <Package_Name> указать название создаваемого пакета. Затем приводятся шаблоны декларации новых типов, констант, функций и процедур, которые можно использовать для объявления всех необходимых элементов создаваемого пакета. В строке, завершающей блок декларации формируемого пакета, нужно после ключевого слова ***end*** заменить <Package_Name> названием этого пакета. Аналогична процедура замены в строках, которые открывают и завершают описание тела пакета, после ключевых слов ***package body*** и ***end*** соответственно. В теле пакета приведены два примера определения функций и пример определения процедуры, которые можно использовать в качестве шаблонов. Для формирования законченного описания VHDL-пакета целесообразно воспользоваться шаблонами встроенного HDL-редактора пакета САПР серии Xilinx ISE Design Suite, механизм применения которых рассматривается в следующем разделе.

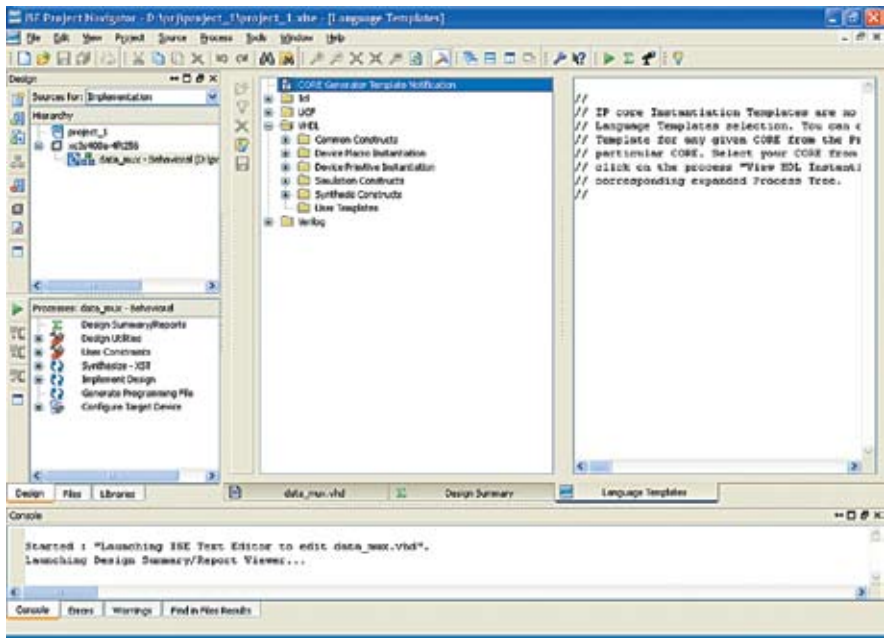

Рис. 5. Навигатор проекта (Project Navigator) после активизации окна шаблонов Language Templates

Использование шаблонов встроенного HDL-редактора САПР серии Xilinx ISE Design Suite

HDL-редактор пакета САПР серии Xilinx ISE Design Suite включает шаблоны для языков описания аппаратуры высокого уровня Verilog и VHDL. Кроме того, имеется группа шаблонов временных и топологических ограничений, предназначенных для формирования файлов UCF (User Constraints File). В последних версиях рассматриваемых средств проектирования предусмотрен также набор шаблонов команд управления.

Чтобы воспользоваться всеми этими шаблонами, нужно прежде всего открыть рабочее окно интегрированного HDL-редактора, в котором создается VHDL-описание разрабатываемого устройства или одного из его модулей. Далее следует активизировать окно шаблонов, используя команду ***Language Templates*** из всплывающего меню ***Edit*** или кнопку . При этом открывается новое окно с заголовком ***Language Templates***, вид которого представлен на рис. 5.

Окно шаблонов содержит две встроенные панели. В левой части окна отображается панель, содержащая список шаблонов, а в правой — панель, в которой открывается текст выбранного шаблона. Первоначально отображаются четыре папки с названиями ***Verilog***, ***VHDL***, ***UCF*** и ***TCL***, в которых содержатся шаблоны соответствующих языков описания аппаратуры HDL, ограничений проекта и команд управления.

Чтобы включить требуемый шаблон в состав создаваемого VHDL-описания, нужно открыть папку VHDL, дважды щелкнув левой кнопкой мыши. После этого в той же встроенной панели будут показаны основные группы шаблонов в виде папок с соответствующими названиями. Краткое описание этих групп шаблонов приводится ниже в данном разделе. В каждой из основных групп шаблоны разделены на подгруппы. Поэтому далее нужно последовательно открыть папки соответствующей группы и подгруппы, пока на экране в левой встроенной панели окна ***Language Templates*** не появится список шаблонов, входящих в состав выбранной подгруппы. В этом списке на соответствующей строке следует кликнуть название требуемого шаблона, после чего текст содержимого выбранного шаблона отобразится в правой встроенной панели. Затем нужно воспользоваться командой ***Use in…*** из всплывающего меню Edit или контекстно-зависимого меню, которое выводится на экран щелчком правой кнопки мыши. Кроме того, для этой же цели можно использовать кнопку быстрого доступа (рис. 5). При этом шаблон будет вставлен в то место создаваемого файла VHDL-

описания, где расположен курсор в рабочем окне интегрированного HDL-редактора. Если требуется вставить только фрагмент шаблона, то необходимо выделить его в правой встроенной панели окна ***Language Templates*** и выполнить команды копирования (***Copy***) и вставки (***Paste***) из всплывающего меню ***Edit*** или контекстно-зависимого меню.

Для языка VHDL в одноименной папке представлены шесть основных групп шаблонов, оформленные в виде папок со следующими названиями: ***Common Constructs***, ***Device Macro Instantiation***, ***Device Primitive Instantiation***, ***Simulation Constructs***, ***Synthesis Constructs*** и ***User Templates*** (рис. 5). В папке ***Common Constructs*** содержатся образцы основных базовых конструкций языка VHDL. В разделе ***Device Macro Instantiation*** представлены макросы, предназначенные для описания специализированных аппаратных блоков кристаллов программируемой логики. Папка ***Device Primitive Instantiation*** включает примеры использования компонентов, которые входят в состав библиотеки примитивов, предоставляемой фирмой Xilinx. Она содержит, в частности, VHDL-описания специализированных ресурсов ПЛИС семейств CPLD (Complex Programmable Logic Device) и FPGA. Шаблоны из папки ***Device Primitive Instantiation*** содержат образцы применения этих библиотечных компонентов, которые сгруппированы в иерархическую систему папок в соответствии с архитектурой ПЛИС, для которых они предназначены, и функциональным назначением примитивов. Все описания, относящиеся к группам ***Device Macro Instantiation*** и ***Device Primitive Instantiation***, поддерживаются средствами синтеза VHDL-кода САПР серии Xilinx ISE Design Suite.

Папка ***Synthesis Constructs*** объединяет блоки кода, большинство из которых предназначено для использования в процессе создания поведенческих, потоковых и смешанных описаний проектируемых устройств. Некоторые шаблоны, находящиеся в данной папке, могут также использоваться в структурных описаниях разрабатываемых устройств. Кроме того, в папке ***Synthesis Constructs*** представлены образцы описания основных функциональных элементов цифровых устройств на языке VHDL. В отличие от законченных VHDL-описаний шаблоны в этой папке не содержат ссылок на используемые библиотеки и пакеты, объявлений сигналов и выражений, представляющих интерфейс объекта. Эти выражения обычно приводятся в форме комментариев. Интерфейсные цепи шаблонов могут быть декларированы в создаваемом описании или как порты, или как сигналы. Все описания в папке ***Synthesis Constructs*** выполнены на базе синтезируемого подмножества языка VHDL [2–5].

В ***Simulation Constructs*** собраны шаблоны базовых конструкций языка VHDL, которые могут потребоваться при описании тестовой системы и тестовых воздействий. Эти описания необходимы для осуществления функционального и полного (временного) моделирования проектируемого устройства.

Шаблоны в папках ***Common Constructs, Synthesis Templates*** и ***Simulation Constructs*** носят обобщенный характер и могут использоваться в проектах, реализуемых на базе ПЛИС различных семейств. Большинство элементов, представленных в папках ***Device Macro Instantiation*** и ***Device Primitive Instantiation***, предназначено для конкретных семейств кристаллов программируемой логики, выпускаемых фирмой Xilinx.

Папка ***User Templates*** предназначена для хранения шаблонов, создаваемых разработчиком. Процедура создания и особенности применения пользовательских шаблонов рассматриваются в следующем разделе.

Шаблоны описаний, создаваемые разработчиком

Папка ***User Templates*** в отличие от рассмотренных выше ***Common Constructs***, ***Device Macro Instantiation***, ***Device Primitive Instantiation***, ***Simulation Constructs*** и ***Synthesis Constructs*** изначально не содержит никаких элементов и предназначена для хранения шаблонов, определяемых разработчиком. В процессе работы над проектом многократно используемые блоки VHDL-описаний целесообразно оформить в виде соответствующих шаблонов.

Для того чтобы приступить к созданию нового шаблона, следует выделить название папки ***User Templates*** в окне шаблонов HDL-редактора пакета САПР серии Xilinx ISE Design Suite, кликнув на соответствующей строке списка. После этого нужно воспользоваться кнопкой быстрого доступа (рис. 5) или командой ***New Template*** из всплывающего меню. В результате выполненных действий в папке ***User Templates*** появится новый элемент ***New Template***, доступный для редактирования. Следует ввести название нового шаблона. Далее нужно активизировать встроенную панель редактирования и ввести текст VHDL-описания. Можно использовать команды копирования (***Copy***) и вставки (***Paste***) из всплывающего меню. После окончания редактирования текста следует сохранить шаблон на диске, используя кнопку (рис. 5).

При большом количестве шаблонов, создаваемых разработчиком, для удобства использования целесообразно внутри папки ***User Templates*** распределить их по группам. Чтобы создать новую группу (раздел в папке ***User Templates***), следует выделить название папки ***User Templates*** в окне шаблонов и воспользоваться кнопкой быстрого доступа или командой ***New Folder***. Перед созданием нового шаблона следует выделить в списке папок раздел, в котором он должен быть расположен.

Следует обратить внимание на то, что в каждом проекте папка ***User Templates*** формируется заново. Поэтому, чтобы в новом проекте были доступны шаблоны, созданные в рамках ранее выполнявшегося проекта, необходимо скопировать файл VHDL_user.xml, который находится в папке ***Templates***, из его рабочего каталога в аналогичную папку рабочего каталога нового проекта. При этом неиспользуемые шаблоны можно удалить кнопкой быстрого доступа.

Шаблоны основных базовых конструкций языка VHDL

Шаблоны основных базовых конструкций языка VHDL расположены в папке Common Constructs. Подробная структура этого раздела изображена на рис. 6. Далее шаблоны языка VHDL будут рассматриваться в той же последовательности, как они расположены в папках соответствующих разделов.

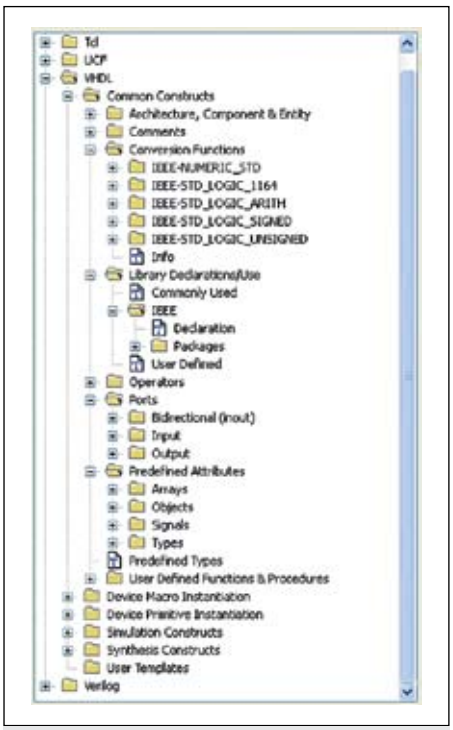

Рис. 6. Структура раздела Common Constructs шаблонов языка VHDL

Базовые конструкции основных структурных элементов языка VHDL

Базовые конструкции основных структурных элементов языка VHDL (описания интерфейса объекта, архитектурного тела, декларации компонента, создания экземпляра компонента) сгруппированы в папке ***Architecture***, ***Component***, ***Entity***.

Architecture Declaration представляет собой основу архитектурного тела описываемого объекта:

```
architecture <arch_name> of <entity_name> is
   -- declarative_items (signal declarations, component declarations, etc.)
begin
   -- architecture body
end <arch_name>;
```

После включения этого шаблона в состав создаваемого файла описания необходимо вместо <*entity_name*> ввести название объекта, а вместо <*arch_name*> — название архитектурного блока описания. Далее нужно заменить строку ***declarative_items*** выражениями декларации сигналов, компонентов, констант, типов, функций и процедур, которые будут использоваться в составе определения архитектуры описываемого объекта. Затем следует вместо строки ***architecture body*** добавить описание структуры или поведения объекта.

Component Declaration представляет собой шаблон объявления компонента, используемого в структурном описании объекта (проекта). Для декларации соответствующего компонента необходимо указать его название (вместо <*component_name*>) и включить описание настраиваемых параметров и интерфейса:

```
component <component_name>
generic (
        <generic_name> : <type> := <value>;
        <other generics>...
);
port (
      <port_name> : <mode> <type>;
      <other ports>...
);
end component;
```

В данном шаблоне нужно заменить <*generic_name*> идентификатором соответствующего настраиваемого параметра, <*type*> — ключевым словом, которое зарезервировано для обозначения требуемого типа, <*value*> — значением, которое должно быть присвоено этому параметру по умолчанию. В каждой строке, содержащей описание интерфейсного порта декларируемого компонента, следует вместо <*port_name*> указать идентификатор соответствующего порта, вместо <*mode*> — режим его работы и вместо <*type*> — обозначение типа.

Component Instantiation содержит образец конструкции, используемой для создания экземпляра компонента в структурном описании объекта (проекта):

```
<instance_name> : <component_name>
generic map (
              <generic_name> => <value>,
              <other generics>...
)
port map (
            <port_name> => <signal_name>,
            <other ports>...
);
```

При формировании конкретного экземпляра компонента следует указать позиционную метку компонента <*instance_name*>, которая соответствует обозначению элемента на принципиальной схеме, название компонента <*component_name*>, которое определяется при создании его описания и соответствует функциональному типу элемента, а также определить требуемые значения настраиваемых параметров и карту портов. В рассматриваемом шаблоне применяется ключевое сопоставление портов компонента и подключаемых сигналов, при котором порядок их перечисления не имеет значения. Поэтому для определения карты портов достаточно вписать идентификатор порта <*port_name*> и название соответствующего сигнала <*signal_name*>, подключаемого к этому порту. Для определения значений настраиваемых параметров нужно указать идентификатор параметра <*generic_name*> и значение этого параметра <*value*> для рассматриваемого экземпляра компонента.

Entity declaration представляет собой основу описания интерфейса объекта. После включения ее в файл следует указать имя объекта (<*entity_name*>) и описание его настраиваемых параметров и интерфейсных портов. Выражения декларации настраиваемых параметров и интерфейсных портов формируются таким же образом, как и при использовании шаблона объявления компонента Component Declaration, рассмотренного выше:

```
entity <entity_name> is
generic (
            <generic_name> : <type> := <value>;
            <other generics>...
);
port (
        <port_name> : <mode> <type>;
        <other ports>...
);
end <entity_name>;
```

Шаблоны комментариев

Шаблоны комментариев расположены в папке ***Comments*** (рис. 6).

Sample File Header содержит блок комментариев, который целесообразно добавить в начало нового создаваемого файла VHDL-описания:

```
----------------------------------------------------
-- Company: <Company Name>
-- Engineer: <Engineer Name>
--
-- Create Date: <date>
-- Design Name: <name_of_top-level_design>
-- Component Name: <name_of_this_component>
-- Target Device: <target device>
-- Tool versions: <tool_versions>
-- Description:
--     <Description here>
-- Dependencies:
-- <Dependencies here>
-- Revision:
--     <Code_revision_information>
-- Additional Comments:
--     <Additional_comments>
----------------------------------------------------
```

В данном блоке можно указать различную информацию, в том числе сведения о разработчике, дату создания, название проекта, для которого предназначен данный модуль, обозначение ПЛИС, на базе которых может быть реализовано описание, номер версии.

Single Line представляет собой шаблон простого однострочного комментария. Для его использования достаточно заменить предлагаемый текст (Comment here) соответствующей строкой:

```
-- Comment here
```

Шаблоны функций преобразования типов данных

В языке VHDL должно строго соблюдаться соответствие типов данных. Поэтому невозможно напрямую присвоить сигналу или переменной значение, которое не согласуется с типом этого сигнала или переменной. Для этой цели нужно использовать функции преобразования типов данных, которые определены в различных пакетах стандартной библиотеки IEEE. Шаблоны функций преобразования типов размещаются в папке ***Conversion Functions*** (рис. 6). Они сгруппированы в соответствии с типом данных, к которому относятся функции преобразования. Каждая группа шаблонов располагается в отдельном каталоге, название которого совпадает с идентификатором пакета библиотеки IEEE, содержащим определения функций. В начале в виде комментариев приведены ссылки на библиотеки и пакеты, в которых дается их определение. При использовании шаблонов нужно поместить указанные ссылки, убрав символы комментариев, в соответствующий раздел создаваемого VHDL-описания, если они не были введены ранее.

В разделе IEEE_NUMERIC_STD находятся шаблоны функций преобразования, относящиеся к различным числовым типам данных. Эти функции определены в пакете numeric_std библиотеки IEEE.

INTEGER to SIGNED представляет собой шаблон использования функции TO_SIGNED, которая осуществляет преобразование данных целого типа INTEGER к знаковому типу SIGNED.

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

Аргументами функции TO_SIGNED являются преобразуемое значение переменной или сигнала целого типа, для которого указывается идентификатор <*int_sig*>, и параметр <*integer_size*>, определяющий длину этого значения, выраженную в количестве бит. Значение рассматриваемой функции присваивается сигналу или переменной знакового типа с идентификатором <*signed_sig*>.

INTEGER to UNSIGNED содержит образец применения функции TO_UNSIGNED, которая 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.numeric_std.all;
--
<unsigned_sig> = TO_UNSIGNED(<int_sig>, <integer_size>);
```

Функция TO_UNSIGNED имеет те же аргументы, что и предыдущая функция преобразования типа. При использовании представленного шаблона следует указать идентифи-

катор сигнала <*unsigned_sig*> или переменную беззнакового типа.

SIGNED to INTEGER включает 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.numeric_std.all;
--
<int_sig> = TO_INTEGER(<signed_sig>);
```

В качестве аргумента функции TO_INTEGER в данном случае нужно указать идентификатор переменной или сигнала знакового типа <*signed_sig*>. Значение функции TO_INTEGER может быть присвоено переменной или сигналу целого типа, для которого указывается идентификатор <*int_sig*>.

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

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

В этом шаблоне в качестве аргумента функции TO_INTEGER должно быть указано имя переменной или сигнала, которые относятся к беззнаковому типу.

В папке IEEE_STD_LOGIC_1164 приведены шаблоны функций преобразования, которые относятся к данным типов STD_ULOGIC и STD_LOGIC. Определения этих функций содержатся в пакете std_logic_1164 библиотеки IEEE.

BIT to STD_ULOGIC содержит шаблон функции To_StdUlogic, предназначенной для приведения значения сигнала или переменной битового типа (BIT) к значению типа STD_ULOGIC.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_1164.all;
--
<sul_sig> = To_StdUlogic(<bit_sig>);
```

Аргументом функции To_StdUlogic является идентификатор переменной или сигнала битового типа (BIT) <*bit_sig*>. Значение этой функции присваивается переменной или сигналу, которые относятся к типу STD_ULOGIC. Указывается имя этой переменной <*sul_sig*>.

BIT VECTOR to STD_LOGIC VECTOR включает совокупность выражений, необходимых для выполнения преобразования массива битового типа (BIT) в массив типа STD_LOGIC VECTOR, которое осуществляется с помощью функции To_StdLogicVector.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_1164.all;
--
<slv_sig> = To_StdLogicVector(<bv_sig>);
```

В последнем выражении, в котором производится обращение к функции To_StdLogicVector, нужно вписать имя массива битового типа <*bv_sig*> и название <*slv_sig*> соответствующего массива типа STD_LOGIC VECTOR.

BIT VECTOR to STD_ULOGIC VECTOR представляет собой шаблон, демонстрирующий применение функции преобразования массива битового типа (BIT) в массив типа STD_ULOGIC VECTOR.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_1164.all;
--
<sulv_sig> = To_StdULogicVector(<bv_sig>);
```

В качестве аргумента функции To_StdULogicVector задается название массива битового типа. Значение этой функции <*sulv_sig*> может быть присвоено массиву типа STD_ULOGIC VECTOR.

STD_LOGIC VECTOR to BIT VECTOR содержит набор выражений, применяемых при обращении к функции To_bitvector, которая предназначена для преобразования массива типа STD_LOGIC VECTOR в массив битового типа (BIT).

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_1164.all;
--
<bv_sig> = To_bitvector(<slv_sig>);
```

При использовании этого шаблона следует в выражении вызова функции To_bitvector указать <*slv_sig*> — имя массива типа STD_LOGIC VECTOR и название битового массива <*bv_sig*>.

STD_LOGIC VECTOR to STD_ULOGIC VECTOR предоставляет разработчику возможность быстрого включения в состав формируемого описания функции To_StdULogicVector, осуществляющей приведение массива подтипа STD_LOGIC VECTOR к массиву основного типа STD_ULOGIC VECTOR.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_1164.all;
--
<sulv_sig> = To_StdULogicVector(<slv_sig>);
```

Аргумент функции To_StdULogicVector задается в виде имени массива <*slv_sig*>, относящегося к подтипу STD_LOGIC VECTOR. Результат преобразования записывается в массив типа STD_ULOGIC VECTOR, указывается название <*sulv_sig*>.

STD_ULOGIC to BIT содержит образец обращения к функции To_bit, которая выполняет преобразование значения переменной или сигнала типа STD_ULOGIC в значение битового типа (BIT).

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_1164.all;
--
<bit_sig> = To_bit(<sul_sig>);
```

При использовании этого шаблона в строке вызова функции To_bit нужно указать идентификатор переменной или сигнала <*sul_sig*> типа STD_ULOGIC и имя переменной или сигнала битового типа <*bit_sig*>.

STD_ULOGIC VECTOR to BIT VECTOR включает совокупность выражений, необходимых для использования функции To_bitvector, осуществляющей конверсию массива типа STD_ULOGIC VECTOR в массив битового типа (BIT).

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_1164.all;
--
<bv_sig> = To_bitvector(<sulv_sig>);
```

В качестве аргумента функции To_bitvector в последней строке шаблона следует указать имя массива типа STD_ULOGIC VECTOR. Результат преобразования <*bv_sig*> может быть записан в массив битового типа.

STD_ULOGIC VECTOR to STD_LOGIC VECTOR представляет собой вариант применения функции To_StdLogicVector для приведения массива типа STD_ULOGIC VECTOR к массиву подтипа STD_LOGIC VECTOR.

```
--The following library will need to be declared for this function:
--library IEEE;
--use IEEE.std_logic_1164.all;
--
<slv_sig> = To_StdLogicVector(<sulv_sig>);
```

В данном варианте использования функции To_StdLogicVector ее аргумент задается в виде имени массива основного типа STD_ULOGIC VECTOR, а для записи результата указывается название массива подтипа STD_LOGIC VECTOR.

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.: Постмаркет, 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.