

# Практический курс сквозного проектирования цифровых устройств на основе ПЛИС фирмы Xilinx

(часть 6)

Валерий Зотов (Москва)

В шестой части курса приводится информация о различных стилях описания проектируемых устройств на языке VHDL. Подробно рассмотрен процесс создания исходных VHDL-описаний разрабатываемого устройства с помощью инструментов САПР серии Xilinx ISE. Приводится краткое описание шаблонов HDL-редактора, входящего в состав данных средств проектирования.

## Последовательно выполняемые операторы языка VHDL

В языке VHDL последовательно выполняемые операторы подобны операторам различных языков программирования высокого уровня, например C/C++. В отличие от параллельных операторов, рассмотренных в предыдущем разделе, порядок взаимного расположения последовательных операторов в определении архитектуры описываемого объекта имеет существенное значение. Последовательные операторы применяются в теле операторов процесса, функций и процедур.

В группу последовательно выполняемых операторов языка VHDL входят:

- последовательный оператор присваивания значения сигнала;
- оператор присваивания значения переменной;
- оператор цикла;
- оператор перехода к следующей итерации цикла;
- оператор принудительного завершения выполнения цикла (выхода из цикла);
- условный оператор;
- оператор множественного выбора;
- оператор ожидания.

В отличие от параллельного оператора назначения сигнала, последовательный оператор используется только в теле процесса. Последовательный оператор назначения сигнала имеет тот же формат, что и аналогичный параллельный оператор, рассмотренный в предыдущем разделе.

Для присвоения значения переменной используется оператор, который имеет следующий формат:

```
<идентификатор_переменной> :=
<выражение, определяющее значение_переменной>;
```

В правой части этого оператора может быть указано как конкретное значение, так и выражение, результат которого должен иметь тот же тип, что и переменная. Например:

```
REG_INIT := 0; SUM := OP1 + OP2;
```

Оператор цикла предназначен для организации циклического повторения некоторой совокупности последовательных операторов определённое или бесконечное число раз. Этот оператор может применяться в одном из трёх возможных вариантов. Первый вариант, используемый для организации бесконечного цикла, имеет следующий формат:

```
[<метка>] : loop
последовательный оператор 1;
последовательный оператор 2;
...
последовательный оператор M;
end loop [<метка>;
```

Началом оператора бесконечного цикла является ключевое слово *loop*, после которого определяется тело цикла в виде совокупности последовательных операторов. Завершает оператор цикла в каждом

варианте сочетание ключевых слов *end loop*.

Второй вариант оператора цикла используется, в основном, в тех случаях, когда количество итераций заранее определено. Синтаксис этого варианта оператора выглядит следующим образом:

```
[<метка>]: for <идентификатор_параметра_цикла> in <начальное_значение_параметра_цикла> to <конечное_значение_параметра_цикла>
loop
последовательный оператор 1;
последовательный оператор 2;
...
последовательный оператор M;
end loop [<метка>;
```

Данный вариант оператора цикла начинается с ключевого слова *for*, после которого указывается идентификатор параметра цикла и диапазон его изменения с помощью ключевых слов *in ... to*. Затем между ключевыми словами *loop* и *end loop* приводятся последовательные операторы, образующие тело цикла.

Третий вариант оператора цикла целесообразно применять в том случае, когда осуществление и завершение итераций цикла зависит от выполнения некоторого условия. При этом используется следующий формат оператора.

```
[<метка>]: while <условное_выражение> loop
последовательный оператор 1;
последовательный оператор 2;
...
последовательный оператор M;
end loop [<метка>;
```

Началом оператора в этом варианте является ключевое слово *while*, за которым приводится выражение, определяющее условие выполнения

цикла. Далее в теле цикла (между ключевыми словами *loop* и *end loop*) записывается необходимая совокупность последовательных операторов. Приведённые ниже примеры иллюстрируют применение трёх вариантов оператора цикла.

```
11 : loop
    SIGN_RE <= '0' after 100 ns;
    SIGN <= '1' after 500 ns;
end loop 11;

12 : for INDX in 0 to 7 loop
    WR(INDX) := MEMR(INDX) + 1;
end loop 12;

13 : while (ADRL < 10) loop
    WMEMR(ADRL) := MEMR(ADRL + 1);
end loop 13;
```

Оператор перехода к следующей итерации цикла позволяет при некотором условии принудительно завершить выполнение текущей итерации и приступить к осуществлению следующей итерации цикла. Формат этого оператора имеет следующий вид:

```
next [<метка_цикла>] when
<условное_выражение>;
```

После ключевого слова *next* может быть указана необязательная метка цикла и условие перехода к следующей итерации. Выражение, определяющее условие перехода, имеет тот же синтаксис, что и в других условных операторах.

Оператор принудительного выхода из цикла завершает выполнение этого цикла и передаёт управление оператору, который следует непосредственно за оператором цикла. Этот переход может выполняться как в условной, так и безусловной форме. Формат оператора выхода из цикла выглядит следующим образом:

```
exit [<метка_цикла>] [when
<условное_выражение>];
```

Основой оператора принудительного завершения цикла является ключевое слово *exit*, после которого в большинстве случаев указывается ключевое слово *when* и условие выхода из цикла. Приведённый ниже пример демонстрирует применение оператора *next* и *exit*.

```
CONTR_SUM:= 0;
J := 0;
lp4 : loop
J := J+1;
next when J<8;
CONTR_SUM := CONTR_SUM + PRM(J);
exit when J>128;
end loop lp4;
```

Условный оператор предназначен для организации исполнения некоторых совокупностей последовательных операторов только при выполнении соответствующих условий. В общем случае применяется следующий формат этого оператора:

```
if <условное_выражение1> then
<группа последовательных операторов1>;
elsif <условное_выражение2> then
<группа последовательных операторов2>;
...
elsif <условное_выражениеN> then
<группа последовательных операторовN>;
else
<группа последовательных операторовN+1>
end if;
```

Началом условного оператора является ключевое слово *if*, после которого указывается условие выполнения первой группы операторов, приведённых после первого ключевого слова *then*. Затем могут следовать несколько выражений *elsif... then*, которые определяют альтернативные условия и соответствующие им совокупности выполняемых операторов. В конце оператора после ключевого слова *else* может быть приведена группа операторов, которая выполняется в том случае, если ни одно из приведённых выше условных выражений не является истинным. Завершением условного оператора является сочетание ключевых слов *end if*. В простейшем варианте условного оператора выражения *elsif... then* и *else* отсутствуют. Использование условного оператора поясняет следующий пример:

```
if PAR < 10 then
R := 1;
elsif PAR < 100 then
R := 2;
elsif PAR < 1000 then
R := 3;
else
```

```
R := 4;
end if;
```

Оператор множественного выбора анализирует значение заданного выражения и, в зависимости от полученного результата, выполняет совокупность операторов, соответствующую этому варианту. Синтаксис этого оператора выглядит следующим образом:

```
case <выражение_выбора> is
    when <значение1_выражения_выбора> =>
        <группа последовательных операторов1>;
    when <значение2_выражения_выбора> =>
        <группа последовательных операторов2>;
    ...
    when <значениеN-1_выражения_выбора> =>
        <группа последовательных операторовN-1>;
    when others =>
        <группа последовательных операторовN>;
end case;
```

Оператор множественного выбора начинается с ключевого слова *case*, после которого указывается выражение выбора. Значение этого выражения определяет, какая из представленных далее групп операторов будет выполняться. Затем следует ключевое слово *is*, после которого приводятся возможные варианты значения выражения выбора и соответствующие им группы операторов. Описание каждого из возможных вариантов начинается с ключевого слова *when*. В заключительной части оператора выбора с помощью сочетания ключевых слов *when others* указывается последовательность операторов, которая выполняется в том случае, когда выражение выбора принимает значение, отличающееся от всех перечисленных выше. Завершается оператор выбора сочетанием ключевых слов *end case*. Например,

```
case SEL_PAR is
when 0 =>
OUT_S <= '0';
when 1 =>
OUT_S <= '1';
when others =>
OUT_S <= 'Z';
end case;
```

Оператор ожидания используется для временного приостановления выполнения процесса или процедуры. Формат этого оператора в общем случае выглядит следующим образом:

```
wait [on <список_чувствительности>] [until <условное_выражение>]
[for <максимальная_длительность_ожидания>];
```

Началом оператора является ключевое слово *wait*, после которого указываются факторы, возобновляющие выполнение процесса. С помощью ключевого слова *on* задаётся список сигналов, изменение которых приводит к активизации процесса. Ключевое слово *until* позволяет определить условие возобновления выполнения процесса. Максимальная длительность состояния ожидания указывается после ключевого слова *for*. При использовании данного оператора в полной форме процесс будет находиться в состоянии ожидания до тех пор, пока не изменится значение одного из сигналов в списке чувствительности и при этом будет выполняться указанное условие. Выполнение процесса возобновляется также, если время ожидания превысило указанное максимальное значение. На практике чаще всего используется сокращённый формат оператора ожидания, когда указывается только один из возможных факторов активизации процесса. Например,

```
wait on CLK, SET_UP;
wait until RES'event and RES='1';
```

### ПРИМЕНЕНИЕ РАЗЛИЧНЫХ СТИЛЕЙ ОПРЕДЕЛЕНИЯ АРХИТЕКТУРЫ ОБЪЕКТА VHDL-ОПИСАНИЯ ПРОЕКТИРУЕМОГО УСТРОЙСТВА

Для определения архитектуры объекта VHDL-описания проектируемого устройства могут использоваться следующие стили:

- структурный (structural description);
- поведенческий (behavioral description);
- потоков данных (dataflow description);
- смешанный.

При структурном методе описания архитектура объекта представляется в виде совокупности взаимосвязанных компонентов. В качестве компо-

нентов могут использоваться VHDL-описания как стандартных элементов цифровых устройств, так и более сложных функциональных модулей разрабатываемого устройства. Основным оператором при использовании данного метода является оператор создания экземпляра компонента (конкретизации компонента). Для описания соединений компонентов используются внутренние сигналы, объявленные в теле архитектуры перед ключевым словом *begin*. Структурное описание подобно схематическому описанию, представленному в текстовой форме записи в терминах языка VHDL. Структурный стиль определения архитектуры объекта целесообразно использовать при создании VHDL-описания верхнего уровня иерархии проекта.

В качестве примера, иллюстрирующего применение структурного метода определения архитектуры разрабатываемого устройства, ниже приводится VHDL-описание дешифратора 3 на 4 *decode3\_4*. Этот дешифратор выполняет преобразование трёхразрядного двоичного кода, подаваемого на входы *din0*, *din1* и *din2*, в сигнал высокого уровня на одном из его выходов, номер которого определяется значением входного кода. При этом двум последовательным значениям входного кода соответствует один выход дешифратора. Например, значениям входного кода 0 и 1 (в десятичной системе счисления) соответствует сигнал высокого уровня на выходе *dout0*, значениям 2 и 3 – сигнал высокого уровня на выходе *dout1*, значениям 4 и 5 – сигнал высокого уровня на выходе *dout2*, значениям 6 и 7 – сигнал высокого уровня на выходе *dout3*.

В приведённом описании дешифратора *decode3\_4* в качестве компонентов использованы предопределённые библиотечные примитивы фирмы Xilinx *and3*, *and3b1*, *and3b2*, *and3b3* и *or2*, которые представляют собой логические элементы 3И без инверсии и с инверсией по одному, двум и трём входам, а также логический элемент ИЛИ.

Поведенческий стиль предполагает описание функционирования объекта в алгоритмической форме. Для этих целей используется, как правило, оператор процесса, последовательно выполняемые операторы, функции и процедуры. Данный стиль позволяет определить поведение

описываемого объекта без конкретизации его внутренней структуры. Применение поведенческого метода при описании проектируемого устройства продемонстрируем на примере рассмотренного выше дешифратора *decode3\_4*. Один из возможных вариантов поведенческого описания этого дешифратора выглядит следующим образом:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--
library UNISIM;
use UNISIM.VComponents.all;
--
entity decode3_4 is
Port (
din0 : in STD_LOGIC;
din1 : in STD_LOGIC;
din2 : in STD_LOGIC;
dout0 : out STD_LOGIC;
dout1 : out STD_LOGIC;
dout2 : out STD_LOGIC;
dout3 : out STD_LOGIC
);
end decode3_4;
--
architecture Behavioral of
decode3_4 is
--
begin
--
process(din0, din1, din2)
variable do0, do1, do2, do3,
do4, do5, do6, do7: STD_LOGIC;
begin
do7 := din0 and din1 and din2;
do6 := not din0 and din1 and
din2;
do5 := din0 and not din1 and
din2;
do4 := not din0 and not din1
and din2;
do3 := din0 and din1 and not
din2;
do2 := not din0 and din1 and
not din2;
do1 := din0 and not din1 and
not din2;
do0 := not din0 and not din1
and not din2;
dout3 <= do6 or do7;
dout2 <= do4 or do5;
dout1 <= do2 or do3;
dout0 <= do0 or do1;
end process;
--
end Behavioral;
```

Основой определения архитектуры Behavioral объекта `decode3_4` в приведённом выше описании является оператор процесса, список чувствительности которого образуют входные сигналы `din0`, `din1` и `din2`. В теле процесса с помощью оператора присваивания значения переменной и стандартных логических операторов языка VHDL выполняются вычисления значений локальных переменных `do0` – `do7`. После этого, используя полученные значения указанных выше локальных переменных, вычисляются значения выходных сигналов `dout0` – `dout3` дешифратора.

Потоковый стиль описания основан на использовании параллельных операторов присваивания значений сигналов (безусловного, условного и выборочного). Описание функционирования разрабатываемого устройства в этом случае представляется в форме совокупности сигналов, образующих некое подобие потоков передачи данных. При этом присваиваемые значения сигналов являются результатами логических и арифметических выражений. Описание дешифратора `decode3_4`, выполненное в потоковом стиле, имеет следующий вид:


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--
library UNISIM;
use UNISIM.VComponents.all;
--
entity decode3_4 is
Port (
din0 : in STD_LOGIC;
din1 : in STD_LOGIC;
din2 : in STD_LOGIC;
dout0 : out STD_LOGIC;
dout1 : out STD_LOGIC;
dout2 : out STD_LOGIC;
dout3 : out STD_LOGIC
);
end decode3_4;
--
architecture Dataflow of
decode3_4 is
signal do0, do1, do2, do3, do4,
do5, do6, do7: STD_LOGIC;
--
begin
--
do7 <= din0 and din1 and din2;
do6 <= not din0 and din1 and din2;
do5 <= din0 and not din1 and din2;
```

```
do4 <= not din0 and not din1 and
din2;
do3 <= din0 and din1 and not din2;
do2 <= not din0 and din1 and not
din2;
do1 <= din0 and not din1 and not
din2;
do0 <= not din0 and not din1 and
not din2;
dout3 <= do6 or do7;
dout2 <= do4 or do5;
dout1 <= do2 or do3;
dout0 <= do0 or do1;
--
end Dataflow;
```

В приведённом потоковом описании архитектуры дешифратора `decode3_4` следует обратить внимание на то, что в отличие от поведенческого стиля описания `do0` – `do7` представляют собой внутренние сигналы, а не локальные переменные.

Между основными тремя стилями определения архитектуры объекта, рассмотренными выше, не существует чётких границ. Поэтому на практике чаще всего используется смешанный метод описания, который представляет собой комбинацию элементов основных стилей.

### СОЗДАНИЕ ОПИСАНИЯ ПРОЕКТИРУЕМОГО УСТРОЙСТВА НА ЯЗЫКЕ VHDL В САПР СЕРИИ XILINX ISE

Процесс создания VHDL-описания разрабатываемого устройства или его функциональных блоков начинается с выполнения процедуры подготовки основы нового исходного модуля проекта, которая активизируется кнопкой  на оперативной панели или командой *New Source* из раздела Project основного меню Навигатора проекта. В качестве типа нового модуля в открывшейся диалоговой панели необходимо выбрать строку *VHDL Module*.

После ввода названия создаваемого модуля и нажатия кнопки Далее (*Next*) открывается диалоговая панель определения исходных данных VHDL-описания, содержащая 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* после его активизации необходимо

указать имя описываемого 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*. Данный идентификатор предполагает, что в создаваемом описании будет использован поведенческий стиль. При необходимости можно изменить данный идентификатор, используя клавиатуру.

Далее следует заполнить таблицу описания портов, которая содержит пять столбцов. Ячейки первого столбца представляют собой поле редактирования, в которое с помощью клавиатуры заносится идентификатор порта *Port Name*. Во второй колонке указывается тип порта *Direction*, который соответствует направлению передачи данных через этот порт. Каждая ячейка данного столбца представляет собой поле выбора, выпадающий список которого содержит три значения, 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, расположенный в столбце *Bus*, в состояние «включено».

Колонки MSB и LSB заполняются только для портов, представленных в виде шин, и портов, описываемых с помощью векторов. В столбце MSB указывается значения индекса, соответствующего старшему разряду вектора, а в LSB – младшему. Если описание портов нового модуля не помещается в видимой части таблицы, следует воспользоваться элементами вертикальной прокрутки, расположенными по правой границе таблицы.

После внесения всех необходимых данных следует нажать кнопку Далее (*Next*), в результате чего открывается панель, в которой отображается вся информация, на базе которой выполняется автоматическое формирование основы нового модуля VHDL-описания. Если все данные, необходимые для создания нового VHDL-описания, указаны корректно, то далее необходимо нажать кнопку Готово (*Finish*) в нижней части информационной па-

нели, в результате чего открывается новое рабочее окно встроенного HDL-редактора, в котором отображается автоматически сформированный код. Этот код включает декларацию используемых библиотек и пакетов, интерфейс описываемого объекта *entity* и основу архитектурного тела VHDL-описания.

Ниже в качестве примера приведён автоматически сформированный текст основы описания устройства, который имеет входы *clk\_s*, *en\_s*, *in\_dat\_bus(7:0)*, *out\_dat\_bus(7:0)*, *out\_s*.

```
-----
-- Company:
-- Engineer:
-- Create Date: 02:36:57
03/12/2007
-- Design Name:
-- Module Name: prim -
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 instanti-
ating
---- any Xilinx primitives in
this code.
--library UNISIM;
--use UNISIM.VComponents.all;
--
entity prim is
Port (
clk_s : in STD_LOGIC;
in_dat_bus : in STD_LOGIC_VECTOR
(7 downto 0);
en_s : in STD_LOGIC;
out_s : in STD_LOGIC;
out_dat_bus : in STD_LOGIC_VEC-
TOR (7 downto 0)
);
end prim;
--
architecture Behavioral of prim is
----
begin
----
end Behavioral;
```

В начале автоматически сформированного текста представлено несколько строк комментариев, в которых можно указать справочную информацию о создаваемом исходном модуле. Далее следуют ссылки на используемую библиотеку IEEE и её 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 её пакет *VComponents*. Данные ссылки необходимо активизировать (убрать символы комментариев) в том случае, если в составе создаваемого описания проектируемого устройства будут использоваться библиотечные компоненты. После ссылок на применяемые библиотеки помещено объявление объекта описания с указанным ранее именем и описание его интерфейса, которое сгенерировано на основе данных о портах устройства, перечисленных в табличной форме.

Завершает текст (автоматически сгенерированной основы модуля VHDL-описания) шаблон блока определения архитектуры описываемого объекта. В этом блоке перед ключевым словом *begin* необходимо поместить выражения декларации всех используемых внутренних сигналов, констант, переменных и компонентов, а также определения функций и процедур, которые будут вызываться при определении архитектуры объекта. После ключевого слова *begin* до заключительной строки *end Behavioral* с помощью параллельно выполняемых операторов необходимо добавить описание функционирования или структуры разрабатываемого устройства, используя один из представленных выше стилей.

Для получения законченного описания на языке VHDL, основу архитектурного тела необходимо дополнить кодом, описывающим внутреннюю структуру или поведение объекта. Текст описания вводится с помощью клавиатуры или шаблонов встроенного HDL-редактора пакета САПР серии Xilinx ISE. Механизм использования шаблонов HDL-описаний рассматривается в последующих разделах.

После завершения формирования модуля текстового описания проекта следует сохранить его в виде файла на диске, используя команды *Save*

или *Save As* из всплывающего меню *File* или кнопку , расположенную на оперативной панели управления Навигатора проекта.

### СОЗДАНИЕ VHDL-ПАКЕТА В САПР СЕРИИ XILINX ISE

Для 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 уже сформированный пакет. В общем случае формат описания VHDL-пакета выглядит следующим образом:


```
[ссылки на используемые библио-
теки и пакеты]
package <название_пакета> is
[декларация новых типов данных]
[декларация сигналов]
[декларация констант]
[декларация компонентов]
[декларация функций]
[декларация процедур]
end <название_пакета>;
package body <название_пакета>
is
[декларация локальных типов дан-
ных]
[декларация локальных констант]
[определения функций]
[определения процедур]
end <название_пакета>;
```

В структуре описания пакета можно выделить три части:

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

Первая часть является необязательной, она присутствует только в тех случаях, когда в описании 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 ключевых слов *package body* и заканчивается ключевым словом *end* с указанием названия пакета. В теле пакета приводятся определения функций и процедур, которые

были объявлены в предыдущей части формируемого 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, с нажатия кнопки  на оперативной панели или активизации команды *New Source* из раздела *Project* основного меню Навигатора проекта. Далее, в открывшейся диалоговой панели *Select Source Type* необходимо выбрать строку *VHDL Package* и в поле редактирования *File Name* указать название создаваемого пакета. После нажатия кнопки Далее (*Next*) в этой диалоговой панели и кнопки Готово (*Finish*) в появившейся вслед за этим информационной панели, автоматически открывается новое рабочее окно интегрированного HDL-редактора, в котором отображается автоматически сформированный шаблон описания пакета. Текст этого шаблона приведён на сайте журнала.

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 ссылками на необходимые библиотеки и пакеты. Далее, после ключевого слова *package* необходимо 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, после ключевого слова *end* необходимо заменить *<Package\_Name>* названием этого пакета. Аналогичную процедуру замены необходимо выполнить 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 один пример определения процедуры, которые можно использовать в качестве шаблонов.


### ИСПОЛЬЗОВАНИЕ ШАБЛОНОВ ВСТРОЕННОГО HDL-РЕДАКТОРА САПР СЕРИИ XILINX ISE

Применение шаблонов, предоставляемых встроенным HDL-редактором САПР серии Xilinx ISE, позволяет:

- сократить время создания исходных описаний проектируемого устройства;
- избежать синтаксических ошибок, появление которых возможно при наборе текста описания с клавиатуры;
- использовать готовые отлаженные конструкции для представления наиболее часто встречающихся элементов и функциональных блоков;
- создавать и использовать в дальнейшем собственные, отработанные конструкции.


HDL-редактор пакета САПР серии Xilinx ISE включает в себя шаблоны для языков ABEL, Verilog и VHDL. Кроме того, имеется группа шаблонов временных и топологических ограничений, предназначенных для формирования файлов UCF (User Constraints File).

Чтобы воспользоваться этими шаблонами, необходимо, прежде всего, открыть (или сделать активным) ра-

бочее окно интегрированного HDL-редактора, в котором создаётся VHDL-описание разрабатываемого устройства или одного из его модулей. Далее следует активизировать окно шаблонов, используя команду *Language Templates* из всплывающего меню *Edit* или кнопку , расположенную на оперативной панели управления Навигатора проекта.

Окно шаблонов содержит две области. В левой области этого окна отображается список шаблонов, которые сгруппированы в папки, а в правой – содержание выбранного шаблона. Первоначально при открытии окна шаблонов в левой области отображаются четыре папки с названиями *ABEL*, *Verilog*, *VHDL* и *UCF*, в которых содержатся шаблоны соответствующих языков описания аппаратуры HDL и ограничений проекта.

Чтобы включить требуемый шаблон в состав создаваемого VHDL-описания, необходимо открыть папку VHDL, поместив курсор на её изображение и дважды щёлкнув левой кнопкой мыши. После этого в той же области окна будут показаны основные группы шаблонов, представленные в виде папок с соответствующими названиями. Краткое описание этих групп шаблонов приводится ниже. В каждой из основных групп шаблоны разделены на подгруппы. Поэтому далее необходимо последовательно открывать папки соответствующей группы и подгруппы, пока на экране в левой области окна не появится список шаблонов, входящих в состав выбранной подгруппы. В этом списке щелчком левой кнопки мыши на соответствующей строке следует выделить название требуемого шаблона, после чего текст выбранного шаблона отобразится в правой области окна.

Затем необходимо воспользоваться командой *Use in...* из всплывающего меню *Edit* или контекстно-зависимого меню, которое выводится на экран щелчком правой кнопки мыши. С этой же целью можно воспользоваться кнопкой быстрого доступа , которая находится на оперативной панели управления Навигатора проекта. При этом шаблон будет вставлен в то место создаваемого файла описания, где расположен курсор. Если требуется вставить только фрагмент шаблона, то необходимо выделить его в правой области окна и выполнить команды копирования и вставки из всплывающего меню *Edit* или контекстно-зависимого меню.

Для языка VHDL в одноименной папке представлены пять основных групп шаблонов, оформленные в виде папок со следующими названиями: *Common Constructs*, *Device Primitive Instantiation*, *Simulation Constructs*, *Synthesis Constructs* и *User Templates*. В папке *Common Constructs* содержатся образцы основных базовых конструкций языка VHDL, которые были рассмотрены в предыдущей части цикла.

Папка *Device Primitive Instantiation* содержит примеры использования компонентов, которые входят в состав библиотеки примитивов, предоставляемой фирмой Xilinx и включающей, помимо прочих, VHDL-описания специализированных ресурсов ПЛИС семейств CPLD и FPGA. Шаблоны, расположенные в папке *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 группе *Device Primitive Instantiation*, поддерживаются средствами синтеза VHDL-кода САПР серии Xilinx ISE.

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 и в структурных описаниях разрабатываемых устройств. Кроме того, в папке *Synthesis Constructs* представлены образцы описания основных функциональных элементов цифровых уст-

ройств на языке VHDL. В отличие от законченных VHDL-описаний, шаблоны, представленные в этой папке, не содержат ссылок на используемые библиотеки и пакеты, объявлений сигналов и выражений, представляющих интерфейс объекта. Эти выражения обычно приводятся в виде комментариев. Интерфейсные цепи шаблонов могут быть декларированы в создаваемом описании или как порты, или как сигналы. Все описания, которые содержатся в папке *Synthesis Constructs*, выполнены на базе синтезируемого подмножества языка VHDL.

В папку *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, сосредоточенные 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 папке *Device Primitive Instantiation*, предназначено для конкретных семейств ПЛИС.


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 папок *Common Constructs*, *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 HDL-описаний целесообразно оформить в виде соответствующих шаблонов.

Чтобы приступить к созданию нового шаблона, следует выделить название папки *User Templates* в окне шаблонов HDL-редактора пакета САПР серии Xilinx ISE, поместив курсор на соответствующей строке списка в левой области и щелкнув левой кнопкой

мышью. После этого необходимо воспользоваться кнопкой быстрого доступа , расположенной на оперативной панели управления Навигатора проекта, или командой *New Template* из всплывающего контекстно-зависимого меню, которое выводится на экран щелчком правой кнопки мыши. В результате выполненных действий в папке *User Templates* появится новый элемент с названием *New Template*, которое доступно для редактирования. Используя клавиатуру, следует ввести название нового шаблона.

Далее необходимо активизировать встроенную панель редактирования в правой области окна шаблонов, расположив на ней курсор и щелкнув левой кнопкой мыши, и с помощью клавиатуры ввести текст HDL-описания. В процессе формирования нового шаблона можно использовать команды копирования и вставки из всплывающего контекстно-зависимого меню. После окончания редактирования текста шаблона следует сохранить его на диске, используя кнопку , которая находится на оперативной панели управления Навигатора проекта.

При большом количестве шаблонов, создаваемых разработчиком, для удобства использования целесообразно распределить их по группам внутри папки *User Templates*. Чтобы создать новую группу (раздел в папке *User Templates*), следует выделить название папки *User Templates* в окне шаблонов и воспользоваться кнопкой быстрого доступа , расположенной на оперативной панели управления Навигатора проекта, или командой *New Folder* из всплывающего контекстно-зависимого меню. Далее, перед созданием нового шаблона, следует выделить в списке папок тот 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 *user\_VHDL.xml*, который находится в рабочем каталоге папки *Templates*, в аналогичную папку рабочего каталога нового проекта. При этом неиспользуемые шаблоны можно удалить, воспользовавшись кнопкой быстрого доступа , расположенной на оперативной панели управления Навигатора проекта.



# Новости мира News of the World Новости мира

## Разрешат копирование HD DVD и Blu-ray дисков

В ближайшем будущем у пользователей должна появиться возможность делать легальные копии фильмов, записанных на диски HD DVD и Blu-ray. Сейчас ведутся переговоры о соглашении, в рамках которого пользователи смогут делать одну резервную копию оригинального диска и ещё одну копию специально для медиа-сервера. Впрочем, сначала нужно, чтобы инициативу AACS Licensing Administrator (Sony, IBM, Walt Disney, Warner Bros. Microsoft) поддержали студии и кинокомпании.

На HD DVD и Blu-ray дисках установлена система защиты AACS (Advanced Access Content System), не позволяющая совершать копирование данных с носителя. Суровая защита является достаточно противоречивым решением, так как ограничивает права пользователей, которым не разрешается перемещать контент на другие цифровые платформы и системы.

От такого решения производители контента должны только выиграть – компании смогут брать дополнительные деньги с пользователей в зависимости от того, сколько копий фильма они собираются сделать.

[forums.vr-zone.com](http://forums.vr-zone.com)

## NEC: результаты и планы на будущее

Закончился очередной финансовый год компании NEC, который предоставил аналитикам пищу для размышлений и выводов. Несмотря на то что рост продаж разработчика составил 7%, показатели прибыльности компании в очередной раз оказались в минусе, т.е. расходы превысили доходы. К примеру, годовые продажи 2006 составили \$5,8 млрд., но в то же время операционный доход оказался отрицательным и составил убыток в 241 млн. долл. Компания сталкивается с отрицательными поквартальными показателями уже довольно давно, начиная с периода апрель-июнь 2005.

В то же время менеджеры NEC Electronics не перестают строить радужных надежд на грядущий год. Аналитики компании утверждают, что в 2007 г. мировой рынок полупроводников вырастет на 5%. Базируясь на этих показателях, NEC надеется повысить уровень влияния на рынке и стабилизировать рост продаж как в полупроводниковом сегменте, так и на всём рынке. К примеру, в 2007 г. компания ожидает роста продаж ЖК-панелей до отметки \$5,7 млрд. Хотя эта сумма всё

же ниже результатов 2006 г., NEC уповает на постепенный рост рынка полупроводников на 3% (до \$5,6 млрд.).

[eetimes.com](http://eetimes.com)

## Galileo требует всё больше денег

Амбициозному европейскому проекту глобальной спутниковой навигационной системы Galileo срочно требуется дополнительное финансирование или проект не будет запущен вовремя. Как заявили официальные лица Европейской комиссии, для успешного ввода в строй системы в 2012 г. проект Galileo нуждается в срочной переработке и дополнительном привлечении порядка 2,4 млрд. евро.

В основе Galileo лежит сеть из 30 спутников, находящихся на орбите на расстоянии 24 тыс. км от земной поверхности. После ввода в строй новая спутниковая система должна стать частью инфраструктуры и полностью покрыть потребность стран Евросоюза в навигационной информации. Кроме того, Galileo предоставит в распоряжение государственных и частных пользователей дополнительные функции: контроль над внешними границами союза, обеспечение транспортной логистики, финансовых операции и операций слежения за критически важными энергетическими и коммуникационными объектами.

Успешный запуск проекта Galileo позволит более чем в два раза увеличить количество рабочих навигационных спутников, доступных пользователям.

[news.zdnet.co.uk](http://news.zdnet.co.uk)

## WUSB-устройства появятся на рынке в середине 2007

Согласно данным аналитического исследования агентства In-Stat, стандартизированные устройства, прежде всего адаптеры и концентраторы с поддержкой беспроводного USB (Wireless USB, WUSB), появятся на рынке уже в середине 2007 г. Основная масса периферийных устройств, таких как принтеры, камеры и переносные компьютеры с интегрированными адаптерами, увидят свет ближе к началу 2008 г.

Первые WUSB-устройства будут предназначены для проведения всевозможных презентаций и выставок, поскольку самое явное преимущество новой технологии – это минимизация количества используемых соединительных кабелей.

Кроме того, общее количество устройств, оснащённых USB, которые были проданы в 2006 г., составило 2 млрд. шт., а ежегодный рост рынка устройств, как со

стандартом USB, так и с беспроводным, будет равняться 12,3% вплоть до 2011 г.

[linuxdevices.com](http://linuxdevices.com)

## Mitsubishi Motors запустит производство Li-ion-батарей

Гибридные автомобили и электромобили – тема, приобретающая всё большую популярность на волне роста озабоченности по поводу сохранения окружающей среды. Автопроизводители задумываются не только о разработке соответствующего транспорта, но и о том, где они будут брать аккумуляторы – один из ключевых и наиболее затратных компонентов этих автомобилей. Недавно стало известно, что совместное предприятие по производству аккумуляторных батарей решили организовать Nissan и NEC, а теперь появились сведения об аналогичной инициативе со стороны компаний Mitsubishi Motors, GS Yuasa и Mitsubishi.



Предприятие будет развёрнуто на территории головного офиса GS Yuasa в Киото и начнёт свою деятельность в ноябре 2007. Стартовый капитал фирмы составит около 25 млн. долл., при этом 51% акций будет принадлежать GS Yuasa, а 34 и 15% распределятся соответственно между Mitsubishi и Mitsubishi Motors. Начальные инвестиции будут потрачены на организацию производственной линии, рассчитанной на выпуск 200 тыс. ионно-литиевых аккумуляторов для автомобилей в год, начало производства которых намечено на 2009 г.

После начала выпуска аккумуляторов на базе совместного предприятия, в 2010 г. у Mitsubishi Motors запланирован выпуск разрабатываемого в настоящее время электромобиля, известного пока что под коротеньким именем i. Ожидается, что на комплектацию i будет отведена половина продукции совместного предприятия – около 100 тыс. аккумуляторов в год, тогда как вторая будет продаваться другим автопроизводителям.

[techon.nikkeibp.co.jp](http://techon.nikkeibp.co.jp)