Продолжение. Начало в № 2`2010

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

Валерий ЗОТОВ
walerry@km.ru

Тридцать вторая часть статьи завершает изучение шаблонов основных базовых конструкций языка VHDL, предназначенных для осуществления моделирования проектируемого устройства. Рассмотрены образцы описаний различных вариантов процессов, запускаемых фронтом сигнала синхронизации, а также шаблоны выражений декларации сигналов, констант, переменных, типов и подтипов. Кроме того, в этой части представлена информация о шаблонах конструкций языка VHDL, используемых в процессе синтеза разрабатываемого устройства. Основное внимание уделено шаблонам записи директив, предназначенных для управления процессами синтеза, отображения логического описания проекта на физические ресурсы ПЛИС (MAP), размещения и трассировки разрабатываемого устройства в кристалле программируемой логики (Place and Route).

В каталоге ***Posedge Clocked***, входящем в состав папки ***Process***, сосредоточены образцы описаний различных вариантов процессов, запускаемых переключением сигнала синхронизации из состояния низкого логического уровня в состояние высокого логического уровня.

Posedge Clocked/w Async & Sync High Reset содержит шаблон описания процесса, активизируемого фронтом тактового сигнала, в котором предоставляется возможность синхронного и асинхронного сброса с активным высоким логическим уровнем сигнала сброса:

```
process (<clock>,<async_reset>)
begin
   if <async_reset> = '1' then
      <statements1>;
   elsif (<clock>'event and <clock> = '1') then
      if <sync_reset> = '1' then
         <statements2>;
      else
         <statements3>;
      end if;
   end if;
end process;
```

При практическом использовании представленной конструкции нужно вместо ***clock*** указать идентификатор тактового сигнала, вместо ***async_reset*** — идентификатор сигнала асинхронного сброса, вместо ***sync_reset*** — идентификатор сигнала синхронного сброса. Совокупность последовательных операторов, выполняемых при запуске процесса и активном уровне сигнала асинхронного сброса, записывается вместо ***statements1***. Последовательность операторов, определяющих поведение модели при активном уровне сигнала синхронного сброса, указывается на месте ***statements2***. Последовательность операторов, исполняемых при запуске процесса и отсутствии активного уровня сигналов сброса, необходимо поместить вместо ***statements3***.

Posedge Clocked/w Async & Sync Low Reset включает в себя образец описания процесса, запускаемого фронтом тактового сигнала, в котором предусмотрена возможность синхронного и асинхронного сброса с активным низким логическим уровнем сигнала сброса:

```
process (<clock>,<async_reset>)
begin
   if <async_reset> = '0' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      if <sync_reset> = '0' then
         <statements>;
      else
         <statements>;
      end if;
   end if;
end process;
```

Posedge Clocked/w Async High Reset представляет собой шаблон описания процесса, активизируемого фронтом тактового сигнала, в котором предусмотрена возможность асинхронного сброса с активным высоким логическим уровнем сигнала сброса:

```
process (<clock>,<reset>)
begin
   if <reset> = '1' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      <statements>;
   end if;
end process;
```

При включении приведенной конструкции в состав создаваемого модуля VHDL-описания необходимо заменить ***clock*** идентификатором сигнала синхронизации, а ***reset*** — идентификатором сигнала асинхронного сброса.

Posedge Clocked/w Async High Reset & CE демонстрирует образец описания процесса, запускаемого фронтом тактового сигнала, в котором предоставляется возможность использования разрешения синхронизации и асинхронного сброса с активным высоким логическим уровнем сигнала сброса:

```
process (<clock>,<reset>)
begin
   if <reset> = '1' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      if <clock_enable> = '1' then
         <statements>;
      end if;
   end if;
end process;
```

При практическом использовании приведенной конструкции в составе формируемого модуля VHDL-описания нужно кроме выполнения указаний, перечисленных при рассмотрении предыдущих шаблонов, заменить ***clock_enable*** идентификатором сигнала разрешения синхронизации.

Posedge Clocked/w Async Low Reset является шаблоном описания процесса, активизируемого фронтом тактового сигнала, в котором предусмотрена возможность асинхронного сброса с активным низким логическим уровнем сигнала сброса:

```
process (<clock>,<reset>)
begin
   if <reset> = '0' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      <statements>;
   end if;
end process;
```

Posedge Clocked/w Async Low Reset & CE содержит образец описания процесса, запускаемого фронтом тактового сигнала, в котором предоставляется возможность использования разрешения синхронизации и асинхронного сброса с активным низким логическим уровнем сигнала сброса:

```
process (<clock>,<reset>)
begin
   if <reset> = '0' then
      <statements>;
   elsif (<clock>'event and <clock> = '1') then
      if <clock_enable> = '1' then
         <statements>;
      end if;
   end if;
end process;
```

Posedge Clocked/w Sync High Reset предоставляет шаблон описания процесса, активизируемого фронтом тактового сигнала, в котором предусмотрена возможность синхронного сброса с активным высоким логическим уровнем сигнала сброса:

```
process (<clock>)
begin
   if (<clock>'event and <clock> = '1'>) then
      if <reset> = '1' then
         <statements>;
      else
         <statements>;
      end if;
   end if;
end process;
```

Posedge Clocked/w Sync Low Reset включает в себя образец описания процесса, запускаемого фронтом тактового сигнала, в котором предоставляется возможность осуществления синхронного сброса с активным низким логическим уровнем сигнала сброса:

```
process (<clock>)
begin
   if (<clock>'event and <clock> = '1'>) then
      if <reset> = '0' then
         <statements>;
      else
         <statements>;
      end if;
   end if;
end process;
```

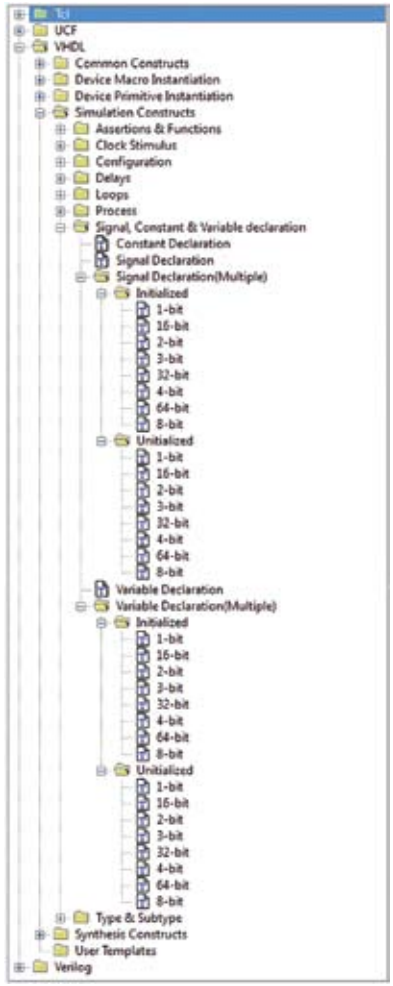

Рис. 521. Структура папки Signal, Constant & Variable Declaration раздела Simulation Constructs шаблонов языка VHDL

Папка ***Signal, Constant & Variable Declaration*** объединяет шаблоны выражений декларации сигналов, констант и переменных. Развернутая структура этой папки показана на рис. 521.

Constant Declaration является шаблоном выражения декларации и инициализации значения константы:

```
constant <name>: <type> := <value>;
```

После включения приведенного выражения в состав создаваемого модуля VHDL-описания нужно заменить ***name*** идентификатором декларируемой константы, а вместо ***type*** и ***value*** указать соответственно тип объявляемой константы и ее значение. ***Signal Declaration*** представляет собой образец обобщенного выражения декларации и инициализации значения сигнала:

```
signal <name>: <type> := <value>;
```

При использовании этого выражения следует вместо ***name*** записать идентификатор объявляемого сигнала, вместо ***type*** — его тип, а вместо ***value*** — присваиваемое начальное значение.

В разделе ***Signal Declaration (Multiple)*** находятся шаблоны выражений декларации сигналов стандартного логического типа (std_logic или std_logic_vector), применяемых для описания шин с различной разрядностью. Объявляемые одноразрядные сигналы относятся к стандартному типу std_logic, а многоразрядные — к типу std_logic_vector. Этот раздел включает в себя два каталога — ***Initialized*** и ***Unitialized*** (рис. 521). В каталоге ***Initialized*** сгруппированы образцы выражений, предназначенных для объявления и инициализации сигналов стандартного логического типа, представленных в форме массивов с различной длиной. В этих шаблонах для инициализации декларируемых сигналов по умолчанию предлагается нулевое значение. При необходимости можно указать требуемое значение в формате, соответствующем разрядности объявляемого сигнала.

Signal Declaration (Multiple)/Initialized/1-bit содержит образец выражения декларации и инициализации одноразрядного сигнала стандартного логического типа:

```
signal <name>: std_logic:= '0';
```

Signal Declaration (Multiple)/Initialized/16-bit включает в себя шаблон выражения объявления и определения начального значения 16-разрядного сигнала (шины) стандартного логического типа:

```
signal <name>: std_logic_vector(15 downto 0):= x"0000";
```

В приведенном выражении начальное значение декларируемого сигнала указывается в виде 4-разрядного шестнадцатеричного числа.

Signal Declaration (Multiple)/Initialized/2-bit демонстрирует образец выражения, применяемого для декларации и инициализации двухразрядного сигнала стандартного логического типа:

```
signal <name>: std_logic_vector(1 downto 0):= "00";
```

В этом выражении начальное значение декларируемого сигнала записывается в виде двухразрядного двоичного числа.

Signal Declaration (Multiple)/Initialized/3-bit предоставляет шаблон выражения, используемого для объявления и определения начального значения трехразрядного сигнала стандартного логического типа:

```
signal <name>: std_logic_vector(2 downto 0):= "000";
```

Исходное значение сигнала в представленном выражении задается в форме трехразрядного двоичного числа.

Signal Declaration (Multiple)/Initialized/32-bit содержит образец выражения декларации и инициализации 32-разрядного сигнала стандартного логического типа:

```
signal <name>: std_logic_vector(31 downto 0):= x"00000000";
```

В этом выражении начальное значение декларируемого сигнала указывается в форме 8-разрядного шестнадцатеричного числа.

Signal Declaration (Multiple)/Initialized/4-bit является шаблоном выражения, применяемого для объявления и определения начального значения 4-разрядного сигнала стандартного логического типа:

```
signal <name>: std_logic_vector(3 downto 0):= "0000";
```

В приведенном шаблоне инициализация декларируемого сигнала осуществляется с помощью 4-разрядного двоичного числа.

Signal Declaration (Multiple)/Initialized/64-bit представляет собой образец выражения декларации и инициализации 64-разрядного сигнала стандартного логического типа:

```
signal <name>: std_logic_vector(63 downto 0):= x"0000000000000000";
```

В этом шаблоне начальное значение объявляемого сигнала задается в форме 16-разрядного шестнадцатеричного числа.

Signal Declaration (Multiple)/Initialized/8-bit включает в себя шаблон выражения, предназначенного для объявления и определения начального значения 8-разрядного сигнала стандартного логического типа:

```
signal <name>: std_logic_vector(7 downto 0):= "00000000";
```

В представленном выражении исходное значение сигнала указывается в виде 8-разрядного двоичного числа.

Каталог ***Unitialized*** раздела ***Signal Declaration (Multiple)*** объединяет шаблоны выражений декларации сигналов стандартного логического типа, представленных в виде массивов соответствующего типа с различной длиной (рис. 521).

Signal Declaration (Multiple)/Unitialized/1-bit, Signal Declaration (Multiple)/Unitialized/2-bit, Signal Declaration (Multiple)/Unitialized/3-bit, Signal Declaration (Multiple)/Unitialized/4-bit, Signal Declaration (Multiple)/Unitialized/8-bit, Signal Declaration (Multiple)/Unitialized/16-bit, Signal Declaration (Multiple)/Unitialized/32-bit, Signal Declaration (Multiple)/Unitialized/64-bit демонстрируют образцы выражений, используемых для объявления соответственно одно-, двух-, трех-, четырех-, восьми-, 16-, 32- и 64-разрядного сигнала стандартного логического типа:

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

Variable Declaration содержит шаблон обобщенного выражения декларации и инициализации переменной:

```
variable <name>: <type> := <value>;
```

После включения этого шаблона в состав формируемого модуля VHDL-описания необходимо вместо ***name*** вставить идентификатор объявляемой переменной, вместо ***type*** — ее тип, а вместо ***value*** — присваиваемое начальное значение.

В разделе ***Variable Declaration (Multiple)*** расположены образцы выражений декларации переменных скалярного и составного типов, которые базируются на стандартном логическом типе (std_logic). Этот раздел разбит на два каталога — ***Initialized*** и ***Unitialized*** (рис. 521). В каталоге ***Initialized*** представлены шаблоны выражений, применяемых для объявления и инициализации переменных стандартного логического типа (std_logic или std_logic_vector), представленных в форме массивов различной длины. Во всех шаблонах этого каталога для определения начального значения декларируемых переменных по умолчанию предлагается нулевое значение, которое можно заменить требуемым числом в соответствующем представлении.

Variable Declaration (Multiple)/Initialized/1-bit является шаблоном выражения декларации переменной стандартного логического типа:

```
variable <name>: std_logic:= '0';
```

Variable Declaration (Multiple)/Initialized/16-bit предоставляет образец выражения, используемого для объявления и определения начального значения 16-разрядной переменной стандартного логического типа:

```
variable <name>: std_logic_vector(15 downto 0):= x"0000";
```

При практическом применении этого шаблона исходное значение декларируемой переменной определяется в виде 4-разрядного шестнадцатеричного числа.

Variable Declaration (Multiple)/Initialized/2-bit включает в себя образец выражения, предназначенного для декларации и инициализации двухразрядной переменной стандартного логического типа:

```
variable <name>: std_logic_vector(1 downto 0):= "00";
```

В представленном выражении начальное значение объявляемой переменной задается в виде двухразрядного двоичного числа.

Variable Declaration (Multiple)/Initialized/3-bit демонстрирует шаблон выражения, применяемого для объявления и определения начального значения трехразрядной переменной стандартного логического типа:

```
variable <name>: std_logic_vector(2 downto 0):= "000";
```

После включения приведенного выражения в состав создаваемого модуля VHDL-описания требуемое начальное значение декларируемой переменной указывается в форме трехразрядного двоичного числа.

Variable Declaration (Multiple)/Initialized/32-bit представляет собой образец выражения, предназначенного для декларации и инициализации 32-разрядной переменной стандартного логического типа:

```
variable <name>: std_logic_vector(31 downto 0):= x"00000000";
```

В приведенном выражении начальное значение объявляемой переменной записывается в виде 8-разрядного шестнадцатеричного числа.

Variable Declaration (Multiple)/Initialized/4-bit содержит шаблон выражения, используемого для объявления и определения начального значения 4-разрядной переменной стандартного логического типа:

```
variable <name>: std_logic_vector(3 downto 0):= "0000";
```

При практическом применении этого выражения можно изменить начальное значение декларируемой переменной, которое указывается в виде 4-разрядного двоичного числа.

Variable Declaration (Multiple)/Initialized/64-bit является образцом выражения, применяемого для декларации и инициализации 64-разрядной переменной стандартного логического типа:

```
variable <name>: std_logic_vector(63 downto 0):= x"0000000000000000";
```

В представленном выражении исходное значение объявляемой переменной задается в виде 16-разрядного шестнадцатеричного числа.

Variable Declaration (Multiple)/Initialized/8-bit демонстрирует шаблон выражения, предназначенного для объявления и определения начального значения 8-разрядной переменной стандартного логического типа:

```
variable <name>: std_logic_vector(7 downto 0):= "00000000";
```

Инициализация декларируемой переменной в этом шаблоне осуществляется в форме 8-разрядного двоичного числа.

В каталоге ***Unitialized*** раздела ***Variable Declaration (Multiple)*** собраны шаблоны выражений декларации переменных с различной разрядностью, относящихся к стандартному логическому типу (std_logic или std_logic_vector).

Variable Declaration (Multiple)/Unitialized/1-bit, Variable Declaration (Multiple)/Unitialized/2-bit, Variable Declaration (Multiple)/Unitialized/3-bit, Variable Declaration (Multiple)/Unitialized/4-bit, Variable Declaration (Multiple)/Unitialized/8-bit, Variable Declaration (Multiple)/Unitialized/16-bit, Variable Declaration (Multiple)/Unitialized/32-bit, Variable Declaration (Multiple)/Unitialized/64-bit предоставляют образцы выражений, используемых для объявления соответственно одно-, двух-, трех-, четырех-, восьми-, 16-, 32- и 64-разрядной переменной стандартного логического типа, имеют вид:

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

В папке ***Type & Subtype*** расположены шаблоны выражений, предназначенных для описания типов и подтипов. Подробная структура этой папки показана на рис. 522.

В состав папки ***Type & Subtype*** входят два раздела — ***Subtype Declaration*** и ***Type Declaration***. В разделе ***Subtype Declaration*** представлены образцы выражений, используемых для описания различных подтипов.

Array включает в себя шаблон выражения определения подтипа, являющегося частью базового типа — массива:

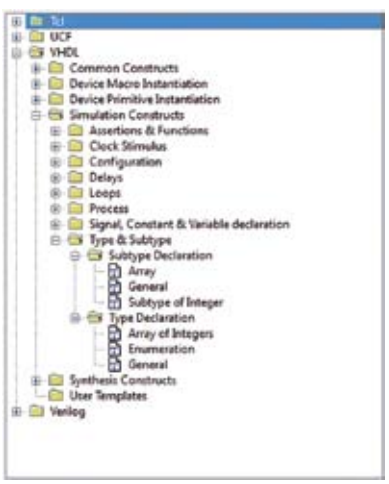

Рис. 522. Структура папки Type & Subtype раздела Simulation Constructs шаблонов языка VHDL

```
subtype <subtype_name> is array range <lower_limit> to <upper_limit>;
```

При практическом использовании этого выражения следует заменить ***subtype_name*** именем декларируемого подтипа, а ***lower_limit*** и ***upper_limit*** — соответственно минимальным и максимальным значением, которое может принимать элемент указанного подтипа.

General является образцом обобщенного выражения, предназначенного для декларации подтипа:

```
subtype <subtype_name> is subtype <subtype_definition>;
```

Subtype of Integer содержит шаблон описания подтипа, представляющего собой подмножество базового целочисленного типа:

```
subtype <subtype_name> is range <lower_limit> to <upper_limit>;
```

Раздел Type Declaration объединяет в себе образцы различных вариантов определения типов данных.

Array of Integers демонстрирует образец выражения, определяющего тип, представляющий собой массив элементов целочисленного типа:

```
type <type_name> is array integer range <lower_limit> to <upper_limit>;
```

После включения приведенного выражения в состав формируемого модуля VHDL-описания нужно вместо ***type_name*** указать имя декларируемого типа, а вместо ***lower_limit*** и ***upper_limit*** — минимальное и максимальное значение соответственно, которое может принимать элемент объявляемого типа.

Enumeration предоставляет шаблон выражения, предназначенного для описания типа, которое осуществляется с использованием списка возможных значений:

```
type <type_name> is (<string1>, <string2>, …);
```

При практическом применении этого шаблона следует заменить <string1>, <string2>,… списком значений, которые может принимать объект определяемого типа.

General включает в себя образец обобщенного выражения, предназначенного для декларации типа:

```
type <type_name> is type <type_definition>;
```

Шаблоны конструкций языка VHDL, используемых в процессе синтеза проектируемого устройства

Шаблоны основных конструкций языка VHDL, предназначенных для управления процессами синтеза и последующего отображения логического описания проекта на физические ресурсы ПЛИС (MAP), размещения и трассировки разрабатываемого устройства в кристалле программируемой логики (Place and Route), находятся в папке ***Synthesis Constructs***. Подробная структура этого раздела представлена на рис. 523.

Рис. 523. Структура раздела Synthesis Constructs шаблонов языка VHDL

В папке ***Assertions & Functions*** расположены шаблоны выражений обращения к процедурам и функциям файлового ввода/вывода, а также образцы применения оператора проверки утверждения и вывода соответствующего сообщения по результатам этой проверки. Эта папка имеет ту же структуру и содержание каталогов, что и одноименная папка, входящая в состав раздела ***Simulation Constructs***, шаблоны которой были рассмотрены в предыдущей части статьи.

Папка ***Attributes*** объединяет в себе образцы использования специальных атрибутов, которые выполняют функцию директив для средств синтеза, отображения синтезированного описания на физические ресурсы ПЛИС и размещения элементов проектируемого устройства в кристаллах программируемой логики. В разделе ***I/O*** этой папки собраны шаблоны конструкций, применяемых для записи директив, управляющих распределением синтезированных элементов в блоках ввода/вывода ПЛИС. В составе этого раздела представлены четыре каталога — ***Don't Place Register Into IOB, I/O Standard, IOB Delay, Place Register Into IOB***. В каталоге ***Don't Place Register Into IOB*** находятся образцы вариантов использования атрибута IOB, запрещающих размещение указанных компонентов и портов в блоках ввода/вывода кристаллов программируемой логики с архитектурой FPGA.

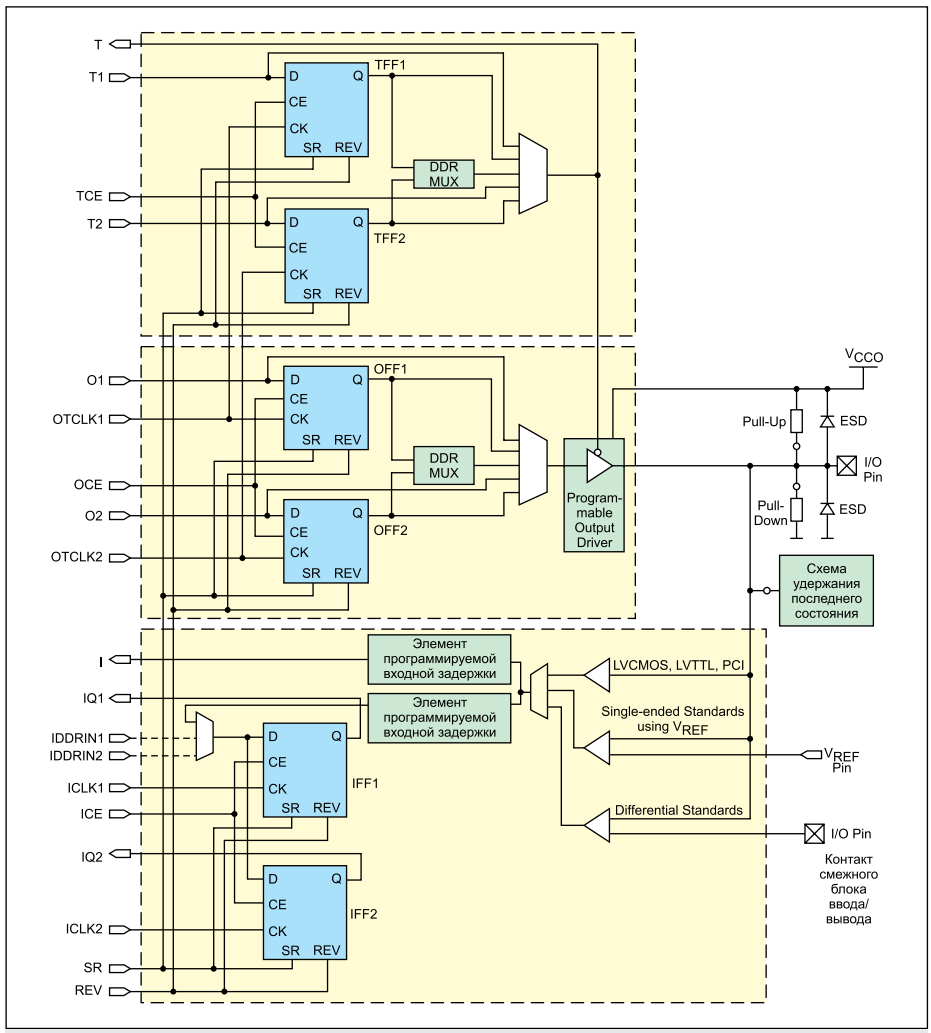

Рис. 524. Структура блоков ввода/вывода ПЛИС семейств Spartan-3, Spartan-3A и Spartan-3E

Component (IOB) содержит шаблон варианта записи атрибута IOB, запрещающего реализацию указанного компонента (триггера или защелки) на базе соответствующих ресурсов блоков ввода/вывода ПЛИС:

```
attribute IOB : string;
attribute IOB of <component_name>: component is "FALSE";
```

При практическом использовании приведенной конструкции необходимо заменить ***component_name*** идентификатором компонента, который не должен располагаться в блоках ввода/вывода кристалла программируемой логики.

Port (IOB) демонстрирует образец варианта применения атрибута IOB, запрещающего размещение заданного порта в составе блока ввода/вывода ПЛИС:

```
attribute IOB : string;
attribute IOB of <port_name>: signal is "FALSE";
```

После включения этой конструкции в состав создаваемого модуля VHDL-описания следует вместо ***port_name*** указать идентификатор соответствующего порта.

В каталоге ***I/O Standard*** представлен шаблон записи атрибута IOSTANDARD, определяющего стандарт ввода/вывода, которому должен соответствовать указанный интерфейсный порт.

Port (IOSTANDARD) является образцом записи атрибута IOSTANDARD, указывающего цифровой сигнальный стандарт, в соответствии с которым должен конфигурироваться вывод кристалла программируемой логики с архитектурой FPGA или семейства CPLD CoolRunner-II, связанный с заданным интерфейсным портом проектируемого устройства:

```
attribute IOSTANDARD : string;
attribute IOSTANDARD of <port_name>: signal is "<standard>";
```

При практическом использовании этого шаблона нужно заменить ***port_name*** идентификатором соответствующего интерфейсного порта, а ***standard*** — условным обозначением требуемого стандарта ввода/вывода. Указываемый цифровой сигнальный стандарт должен поддерживаться ПЛИС семейства FPGA или CPLD CoolRunner-II, используемой для реализации формируемого модуля VHDL-описания.

В каталоге ***IOB Delay*** расположены шаблоны различных вариантов записи директив, предназначенных для управления использованием элементов входной задержки в блоках ввода/вывода кристаллов программируемой логики с архитектурой FPGA. Этот каталог разбит на три подкаталога — ***Component***, ***No Delay*** и ***Spartan-3E/3A***. Подкаталог ***Component*** объединяет образцы различных вариантов использования атрибута IOBDELAY при размещении синтезированных элементов в блоках ввода/вывода.

Both (IOBDELAY) включает в себя шаблон варианта записи атрибута IOBDELAY, допускающего использование элементов входной задержки как в цепи буферного элемента, так и в цепи триггера, представленных в составе блоков ввода/вывода кристаллов программируемой логики с архитектурой FPGA. На рис. 524 изображена структура блоков ввода/вывода ПЛИС семейств Spartan-3, Spartan-3A и Spartan-3E, которая наглядно поясняет предназначение атрибута ***IOBDELAY***:

```
attribute IOBDELAY : string;
attribute IOBDELAY of <component_name>: component is "BOTH";
```

IBUF (IOBDELAY) предоставляет образец варианта использования атрибута IOBDELAY, разрешающего установку элементов входной задержки только в цепи буферного элемента блока ввода/вывода:

```
attribute IOBDELAY : string;
attribute IOBDELAY of <component_name>: component is "IBUF";
```

IFD (IOBDELAY) содержит шаблон варианта записи атрибута IOBDELAY, позволяющего задействовать элементы входной задержки только в цепи триггера, входящего в состав блока ввода/вывода:

```
attribute IOBDELAY : string;
attribute IOBDELAY of <component_name>: component is "IFD";
```

None (IOBDELAY) демонстрирует образец варианта применения атрибута IOBDELAY, запрещающего использование элементов входной задержки при реализации указанного компонента на базе соответствующих ресурсов блока ввода/вывода:

```
attribute IOBDELAY : string;
attribute IOBDELAY of <component_name>: component is "NONE";
```

В подкаталоге ***No Delay*** собраны шаблоны различных вариантов записи директивы NODELAY, которая позволяет исключить элементы входной задержки, представленные в составе блоков ввода/вывода ПЛИС семейств Spartan-3, Spartan-3A и Spartan-3E (рис. 524).

Component (NODELAY) представляет собой образец варианта применения атрибута NODELAY, запрещающего использование элемента входной задержки в составе указан-

ного компонента, реализуемого на базе соответствующих ресурсов блоков ввода/вывода в кристаллах программируемой логики указанных семейств:

```
attribute NODELAY : string;
attribute NODELAY of <component_name>: component is "TRUE";
```

Signal (NODELAY) является шаблоном варианта записи атрибута NODELAY, исключающего элемент входной задержки в цепи указанного сигнала:

```
attribute NODELAY : string;
attribute NODELAY of <signal_name>: signal is "TRUE";
```

В подкаталоге ***Spartan-3E/3A*** находятся образцы использования атрибутов, определяющих величину входной задержки сигнала в цепи буферного элемента или триггера, входящих в состав блоков ввода/вывода ПЛИС семейств Spartan-3E и Spartan-3A.

Non-registered path (IBUF_DELAY_VALUE) содержит шаблон записи атрибута IBUF_DELAY_VALUE, устанавливающего значение входной задержки сигнала в цепи буферного элемента, представленного в составе блока ввода/вывода кристаллов программируемой логики семейств Spartan-3E и Spartan-3A:

```
attribute IBUF_DELAY_VALUE : string;
attribute IBUF_DELAY_VALUE of <port_name>: signal is "<number_
from_0_to_16>";
```

После включения приведенной конструкции в состав формируемого модуля VHDL-описания следует заменить ***port_name*** идентификатором соответствующего порта, а ***number_from_0_to_16*** — целым числом в диапазоне от 0 до 16, значение которого определяет величину входной задержки сигнала в цепи буферного элемента.

Registered path (IFD_DELAY_VALUE) предоставляет образец применения атрибута IFD_DELAY_VALUE для выбора значения входной задержки сигнала в цепи триггера, входящего в состав блока ввода/вывода ПЛИС семейств Spartan-3E и Spartan-3A:

```
attribute IFD_DELAY_VALUE : string;
attribute IFD_DELAY_VALUE of <port_name>: signal is "<number_
from_0_to_8>";
```

При использовании этой конструкции требуемая величина входной задержки задается в виде целого числа, которое не должно выходить за пределы диапазона от 0 до 8.

Каталог ***Place Register Into IOB*** содержит шаблоны вариантов использования атрибута IOB, разрешающих размещение указанных компонентов и портов в блоках ввода/вывода кристаллов программируемой логики с архитектурой FPGA.

Component (IOB) включает в себя образец варианта записи атрибута IOB, допускающего реализацию указанного компонента

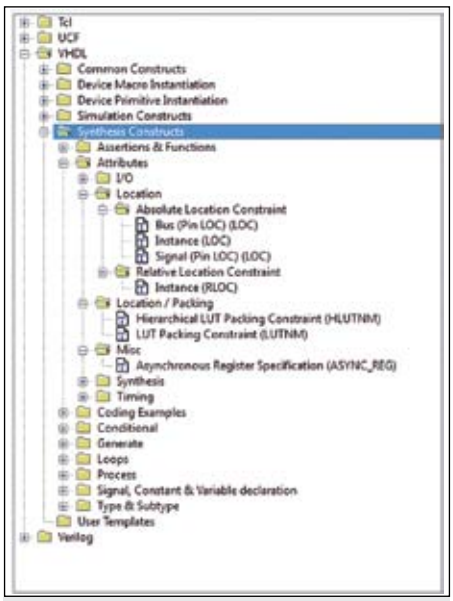

Рис. 525. Структура раздела Location папки Attributes

(триггера или защелки) на базе соответствующих ресурсов блоков ввода/вывода ПЛИС семейств FPGA:

```
attribute IOB : string;
attribute IOB of <component_name>: component is "TRUE";
```

Port (IOB) является шаблоном варианта записи атрибута IOB, разрешающего размещение заданного порта в составе блока ввода/вывода кристалла программируемой логики с архитектурой FPGA:

```
attribute IOB : string;
attribute IOB of <port_name>: signal is "TRUE";
```

В разделе ***Location*** папки ***Attributes*** сгруппированы образцы конструкций, применяемых для записи директив, управляющих размещением синтезированных элементов и цепей в структуре ресурсов ПЛИС. Подробная структура этого раздела показана на рис. 525.

Раздел ***Location*** разбит на два каталога — ***Absolute Location Constraint*** и ***Relative Location Constraint***. В каталоге ***Absolute Location Constraint*** расположены шаблоны вариантов использования атрибута LOC, осуществляющих привязку компонентов и цепей проектируемого устройства к конкретным ресурсам кристалла программируемой логики.

Bus (Pin LOC) (LOC) представляет собой образец варианта записи атрибута LOC, определяющего требуемое расположение указанной шины в структуре ресурсов ПЛИС:

```
attribute LOC : string;
attribute LOC of <bus_name>: signal is "<value>… <value>";
```

При практическом использовании приведенной конструкции нужно вместо ***bus_name*** вставить идентификатор шины, а вместо ***value…value*** указать требуемое размещение проводников этой шины.

Instance (LOC) содержит шаблон варианта записи атрибута LOC, указывающего на требуемое размещение конкретного экземпляра компонента проектируемого устройства в кристалле программируемой логики:

```
attribute LOC : string;
attribute LOC of <inst_name>: label is "<value>";
```

После включения этого шаблона в состав создаваемого VHDL-описания необходимо заменить ***inst_name*** идентификатором привязываемого экземпляра компонента, а ***value*** — условным обозначением расположения соответствующего ресурса ПЛИС.

Signal (Pin LOC) (LOC) демонстрирует образец варианта записи атрибута LOC, определяющего необходимое расположение указанной цепи (сигнала) в кристалле программируемой логики, выбранном для реализации разрабатываемого устройства:

```
attribute LOC : string;
attribute LOC of <signal_name>: signal is "<value>";
```

В приведенной конструкции вместо ***signal_name*** следует вставить идентификатор закрепляемой цепи (сигнала).

В каталоге ***Relative Location Constraint*** находится шаблон записи директивы относительного размещения компонентов проектируемого устройства в ПЛИС.

Instance (RLOC) предоставляет образец применения атрибута RLOC, определяющего относительное положение указанного экземпляра компонента в кристалле программируемой логики:

```
attribute RLOC : string;
attribute RLOC of <inst_name>: label is "<value>";
```

При практическом использовании приведенной конструкции нужно заменить ***inst_name*** идентификатором экземпляра компонента, а ***value*** — условным обозначением относительного расположения этого экземпляра.

Раздел ***Location/Packing*** папки ***Attributes*** объединяет в себе шаблоны конструкций, применяемых для записи директив упаковки пятивходовых таблиц преобразования LUT ПЛИС серии Virtex-5 в шестивходовые функциональные генераторы.

Hierarchical LUT Packing Constraint (HLUTNM) включает в себя образец применения атрибута HLUTNM для создания логической группы из двух пятивходовых таблиц преобразования LUT, относящихся к одному уровню иерархии проекта:

```
-- Specifies LUT packing of two LUT5s into the same LUT6 for Virtex-5
uniquified by hierarchy
attribute HLUTNM : string;
attribute HLUTNM of <inst_name>: label is "<value>";
```

В этой конструкции вместо ***inst_name*** следует вставить идентификатор соответствующего экземпляра таблицы преобразования, а вместо ***value*** — обозначение создаваемой логической группы.

LUT Packing Constraint (LUTNM) представляет собой шаблон записи атрибута LUTNM, осуществляющего компоновку двух пятивходовых таблиц преобразования LUT в шестивходовый функциональный генератор:

```
-- Specifies LUT packing of two LUT5s into the same LUT6 for Virtex-5
attribute LUTNM : string;
attribute LUTNM of <inst_name>: label is "<value>";
```

В разделе ***Misc*** папки ***Attributes*** находится образец записи атрибута ASYNC_REG.

Asynchronous Register Specification (ASYNC_REG) содержит шаблон варианта записи атрибута ASYNC_REG, обеспечивающего корректную обработку асинхронных данных на входах указанного компонента, в том числе в процессе моделирования:

```
attribute ASYNC_REG : string;
attribute ASYNC_REG of <instance_name>: signal is "TRUE";
```

При практическом использовании этого шаблона следует заменить ***instance_name*** идентификатором соответствующего экземпляра триггера или защелки с асинхронными входами.

Раздел ***Synthesis*** предоставляет образцы использования директив, предназначенных для управления процессом синтеза различных элементов формируемого описания разрабатываемого устройства. Детализированная структура этого раздела приведена на рис. 526.

Все шаблоны раздела ***Synthesis*** распределены в 11 каталогов в соответствии с функциональным назначением. В каталоге ***Buffer Type*** расположены образцы различных вариантов записи одноименного атрибута, который предоставляет возможность выбора типа буферного элемента, автоматически добавляемого средствами синтеза во входной или внутренней цепи.

AUTO (BUFFER_TYPE) является шаблоном варианта записи атрибута BUFFER_TYPE, допускающего автоматический выбор типа буферного элемента, включаемого в состав указанной цепи:

```
attribute BUFFER_TYPE : string;
attribute BUFFER_TYPE of <signal_name>: signal is "AUTO";
```

После включения этой конструкции в состав создаваемого VHDL-описания нужно заменить ***signal_name*** идентификатором цепи (сигнала), к которой относится рассматриваемый атрибут.

BUFGDLL (BUFFER_TYPE) демонстрирует образец варианта записи атрибута BUFFER_TYPE, предписывающего автоматическую установку глобального буферного элемента BUFGDLL в заданной цепи:

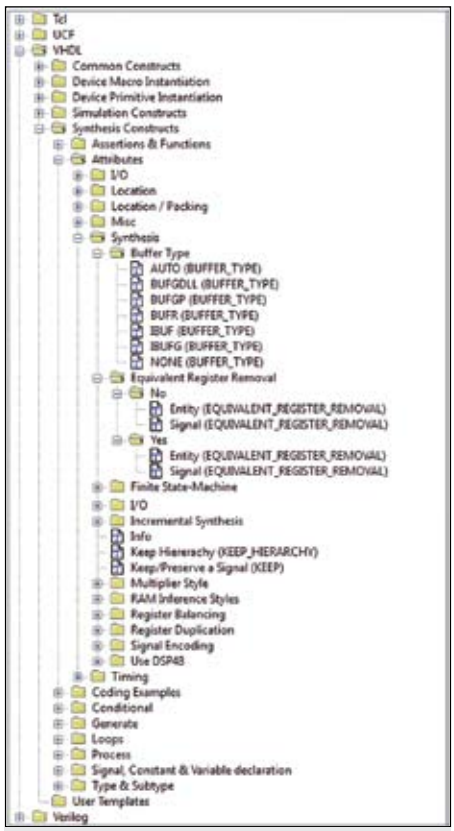

Рис. 526. Структура раздела Synthesis папки Attributes

```
attribute BUFFER_TYPE : string;
attribute BUFFER_TYPE of <signal_name>: signal is "BUFGDLL";
```

BUFGP (BUFFER_TYPE) включает в себя шаблон варианта записи атрибута BUFFER_TYPE, разрешающего автоматическое добавление глобального буферного элемента BUFGP в состав указанной цепи:

```
attribute BUFFER_TYPE : string;
attribute BUFFER_TYPE of <signal_name>: signal is "BUFGP";
```

BUFR (BUFFER_TYPE) предоставляет образец варианта применения атрибута BUFFER_TYPE для автоматической установки регионального буферного элемента BUFR в заданной цепи:

```
attribute BUFFER_TYPE : string;
attribute BUFFER_TYPE of <signal_name>: signal is "BUFR";
```

IBUF (BUFFER_TYPE) содержит шаблон варианта записи атрибута BUFFER_TYPE, указывающего средствам синтеза необходимость автоматического включения входного буферного элемента IBUF в состав соответствующей цепи:

```
attribute BUFFER_TYPE : string;
attribute BUFFER_TYPE of <signal_name>: signal is "IBUF";
```

IBUFG (BUFFER_TYPE) является образцом использования атрибута BUFFER_TYPE для автоматического присоединения входного глобального буферного элемента IBUFG к указанной цепи:

```
attribute BUFFER_TYPE : string;
attribute BUFFER_TYPE of <signal_name>: signal is "IBUFG";
```

NONE (BUFFER_TYPE) представляет собой шаблон варианта записи атрибута BUFFER_TYPE, запрещающего средствам синтеза автоматическую установку буферного элемента в цепи указанного сигнала:

```
attribute BUFFER_TYPE : string;
attribute BUFFER_TYPE of <signal_name>: signal is "NONE";
```

В каталоге ***Equivalent Register Removal*** собраны образцы различных вариантов применения одноименной директивы, предназначенной для управления оптимизацией триггеров (регистров) в процессе синтеза проектируемого устройства. В состав этого каталога входят два подкаталога — ***NO*** и ***YES*** (рис. 526). В подкаталоге ***NO*** находятся шаблоны вариантов записи директивы EQUIVALENT_REGISTER_REMOVAL, запрещающих удаление триггеров, выполняющих эквивалентные функции, а также триггеров, входные сигналы которых имеют постоянный уровень, не изменяющийся в процессе функционирования устройства.

NO/Entity (EQUIVALENT_REGISTER_REMOVAL) демонстрирует образец варианта записи атрибута EQUIVALENT_REGISTER_REMOVAL, не допускающего исключение из состава описания указанного объекта (entity) триггеров, осуществляющих аналогичные функции, и триггеров с постоянными значениями входных сигналов:

```
attribute EQUIVALENT_REGISTER_REMOVAL : string;
attribute EQUIVALENT_REGISTER_REMOVAL of <entity_name>:
entity is "NO";
```

При практическом применении этого шаблона следует вместо ***entity_name*** вставить идентификатор соответствующего объекта описания.

NO/Signal (EQUIVALENT_REGISTER_REMOVAL) включает в себя шаблон варианта записи атрибута EQUIVALENT_REGISTER_REMOVAL, запрещающего удаление триггеров, выполняющих эквивалентные функции, в цепи указанного сигнала:

```
attribute EQUIVALENT_REGISTER_REMOVAL : string;
attribute EQUIVALENT_REGISTER_REMOVAL of <signal_name>:
signal is "NO";
```

В приведенной конструкции необходимо заменить ***signal_name*** идентификатором сигнала, к которому относится этот атрибут.

В подкаталоге ***YES*** находятся образцы использования директивы EQUIVALENT_REGISTER_REMOVAL для исключения в процессе оптимизации триггеров, осуществляющих эквивалентные функции, а также

триггеров с постоянными входными сигналами.

YES/Entity (EQUIVALENT_REGISTER_REMOVAL) предоставляет шаблон варианта записи атрибута EQUIVALENT_REGISTER_REMOVAL, разрешающего удаление из состава описания указанного объекта (entity) триггеров, выполняющих аналогичные функции, и триггеров, входные сигналы которых имеют постоянный уровень:

```
attribute EQUIVALENT_REGISTER_REMOVAL : string;
attribute EQUIVALENT_REGISTER_REMOVAL of <entity_name>: entity is "YES";
```

YES/Signal (EQUIVALENT_REGISTER_REMOVAL) является образцом варианта применения атрибута EQUIVALENT_REGISTER_REMOVAL, предписывающего исключение триггеров, осуществляющих эквивалентные функции, в цепи указанного сигнала:

```
attribute EQUIVALENT_REGISTER_REMOVAL : string;
attribute EQUIVALENT_REGISTER_REMOVAL of <signal_name>: signal is "YES";
```

В каталоге ***Finite State Machine*** раздела ***Synthesis*** размещаются шаблоны конструкций, применяемых для записи директив, предназначенных для управления выбором метода кодирования конечных автоматов в процессе синтеза разрабатываемого устройства. Подробная структура этого каталога показана на рис. 527.

Все шаблоны каталога ***Finite State Machine*** сгруппированы в два подкаталога — ***FSM Encoding*** и ***FSM Extraction***, которые, в свою очередь, разделены на подкаталоги следующего уровня. Подкаталог ***FSM Encoding*** включает в себя подкаталоги ***Compact***, ***Gray***, ***Johnson***, ***One-Hot***, ***Sequential*** и ***User***, каждый из которых предоставляет образцы различных вариантов записи директивы FSM_ENCODING, соответствующих определенному методу кодирования конечных автоматов. В подкаталоге ***Compact*** находятся шаблоны вариантов записи директивы FSM_ENCODING, позволяющих выбрать одноименный алгоритм кодирования. Этот алгоритм позволяет минимизировать количество триггеров при синтезе конечного автомата.

Compact/Entity (FSM_ENCODING) содержит образец варианта записи атрибута FSM_ENCODING, устанавливающего алгоритм кодирования ***Compact*** для указанного объекта описания:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <entity_name>: entity is "COMPACT";
```

Compact/Signal (FSM_ENCODING) представляет собой шаблон варианта записи атрибута FSM_ENCODING, определяющего в качестве метода кодирования конечного автомата алгоритм ***Compact*** для заданного сигнала:

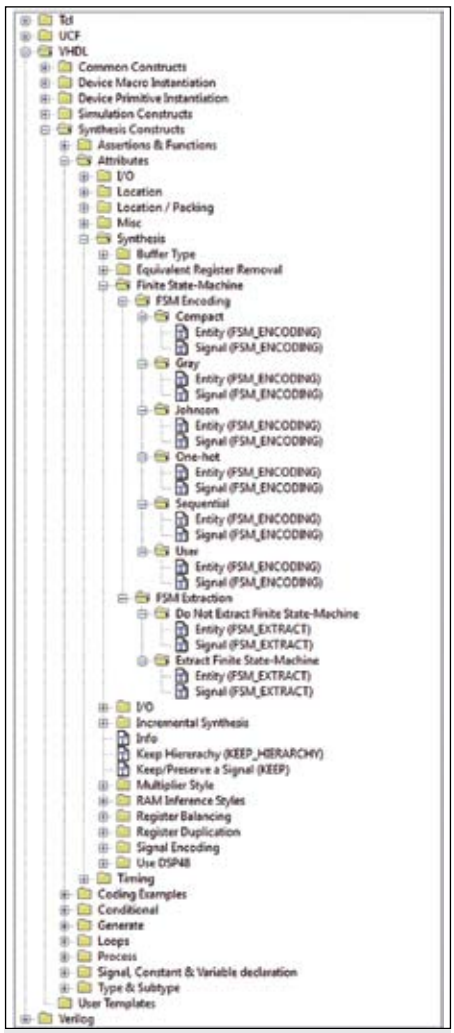

Рис. 527. Структура каталога Finite State Machine раздела Synthesis папки Attributes

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <signal_name>: signal is "COMPACT";
```

В подкаталоге ***Gray*** расположены образцы применения директивы FSM_ENCODING для использования одноименного метода кодирования при синтезе конечных автоматов. Алгоритм ***Gray***, гарантирующий переключение только одной переменной между двумя последовательными состояниями, минимизирует риск возникновения паразитных импульсов.

Gray/Entity (FSM_ENCODING) включает в себя шаблон варианта записи атрибута FSM_ENCODING, устанавливающего метод Gray в качестве алгоритма кодирования конечного автомата для указанного объекта описания:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <entity_name>: entity is "GRAY";
```

Gray/Signal (FSM_ENCODING) демонстрирует образец варианта использования атрибута FSM_ENCODING для выбора метода кодирования конечных автоматов Gray для соответствующего сигнала:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <signal_name>: signal is "GRAY";
```

В подкаталоге ***Johnson*** размещены шаблоны вариантов записи директивы FSM_ENCODING, предписывающих применение одноименного алгоритма кодирования при синтезе конечных автоматов. Метод Johnson целесообразно использовать при синтезе конечных автоматов, описания которых содержат длинные цепочки без ветвлений.

Johnson/Entity (FSM_ENCODING) предоставляет образец варианта применения атрибута FSM_ENCODING для выбора метода кодирования конечных автоматов Johnson при синтезе заданного объекта описания:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <entity_name>: entity is "JOHNSON";
```

Johnson/Signal (FSM_ENCODING) является шаблоном варианта записи атрибута FSM_ENCODING, определяющего в качестве метода кодирования конечного автомата алгоритм Johnson для указанного сигнала:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <signal_name>: signal is "JOHNSON";
```

Подкаталог ***One-Hot*** объединяет в себе образцы использования директивы FSM_ENCODING для выбора одноименного метода кодирования при синтезе конечных автоматов. Метод One-Hot гарантирует, что каждый отдельный регистр предназначен для реализации одного состояния, то есть в любой момент времени активен только один триггер.

One-Hot/Entity (FSM_ENCODING) содержит шаблон варианта записи атрибута FSM_ENCODING, устанавливающего метод One-Hot в качестве алгоритма кодирования конечного автомата для заданного объекта описания:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <entity_name>: entity is "ONE-HOT";
```

One-Hot/Signal (FSM_ENCODING) представляет собой образец варианта применения атрибута FSM_ENCODING для выбора метода кодирования конечных автоматов One-Hot по отношению к указанному сигналу:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <signal_name>: signal is "ONE-HOT";
```

В подкаталоге ***Sequential*** находятся шаблоны вариантов записи директивы FSM_ENCODING, предназначенных для выбора одноименного метода кодирования при синтезе конечных автоматов. Алгоритм Sequential базируется на идентификации длинных ветвей и применении последовательности двоичных кодов для представления состояний этих ветвей.

Sequential/Entity (FSM_ENCODING) включает в себя образец варианта использования атрибута FSM_ENCODING для выбора метода кодирования конечных автоматов Sequential в процессе синтеза соответствующего объекта описания:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <entity_name>: entity is "SEQUENTIAL";
```

Sequential/Signal (FSM_ENCODING) демонстрирует шаблон варианта записи атрибута FSM_ENCODING, определяющего в качестве метода кодирования конечного автомата алгоритм Sequential для заданного сигнала:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <signal_name>: signal is "SEQUENTIAL";
```

В подкаталоге ***User*** расположены образцы вариантов записи директивы FSM_ENCODING, предписывающих средствам синтеза использовать алгоритм кодирования, представленный в файле исходного описания.

User/Entity (FSM_ENCODING) предоставляет шаблон варианта записи атрибута FSM_ENCODING, разрешающего при синтезе указанного объекта описания в качестве метода кодирования конечного автомата применение пользовательского алгоритма:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <entity_name>: entity is "USER";
```

User/Signal (FSM_ENCODING) является образцом варианта применения атрибута FSM_ENCODING для выбора пользовательского метода кодирования конечных автоматов по отношению к указанному сигналу:

```
attribute FSM_ENCODING : string;
attribute FSM_ENCODING of <signal_name>: signal is "USER";
```

В подкаталоге ***FSM Extraction***, входящем в состав каталога ***Finite State Machine***, представлены образцы различных вариантов использования директивы FSM_EXTRACT, которая предназначена для управления извлечением макросов конечных автоматов в процессе синтеза проектируемых устройств. Эти шаблоны распределены по двум подкаталогам — ***Do Not Extract Finite State Machine*** и ***Extract Finite State Machine***. В подкаталоге ***Do Not Extract Finite State Machine*** собраны образцы вариантов записи директивы FSM_EXTRACT, запрещающих выделение макросов конечных автоматов.

Do Not Extract Finite State Machine/Entity (FSM_EXTRACT) содержит шаблон варианта записи атрибута FSM_EXTRACT, не допускающего извлечение макросов конечных автоматов в процессе синтеза указанного объекта описания:

```
attribute FSM_EXTRACT : string;
attribute FSM_EXTRACT of <entity_name>: entity is "NO";
```

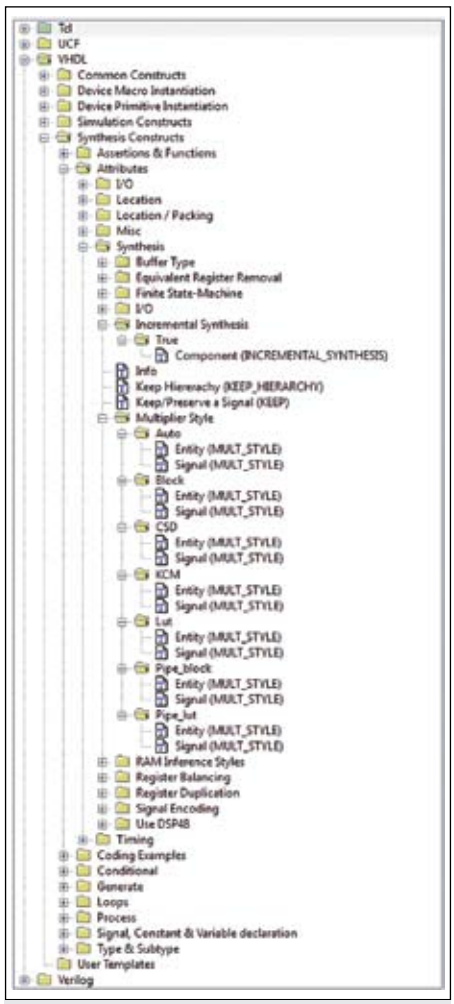

Рис. 528. Структура каталогов Incremental Synthesis и Multiplier Style раздела Synthesis папки Attributes

Do Not Extract Finite State Machine/Signal (FSM_EXTRACT) представляет собой образец варианта применения атрибута FSM_EXTRACT для запрета выделения макросов конечных автоматов в процессе синтеза по отношению к соответствующему сигналу:

```
attribute FSM_EXTRACT : string;
attribute FSM_EXTRACT of <signal_name>: signal is "NO";
```

В подкаталоге ***Extract Finite State Machine*** расположены шаблоны вариантов записи директивы FSM_EXTRACT, разрешающих извлечение макросов конечных автоматов при синтезе разрабатываемых устройств.

Extract Finite State Machine/Entity (FSM_EXTRACT) включает в себя образец использования атрибута FSM_EXTRACT для разрешения выделения макросов конечных автоматов в процессе синтеза заданного объекта описания:

```
attribute FSM_EXTRACT : string;
attribute FSM_EXTRACT of <entity_name>: entity is "YES";
```

Extract Finite State Machine/Signal (FSM_EXTRACT) демонстрирует шаблон варианта записи атрибута FSM_EXTRACT, предписывающего извлечение макросов конечных автоматов при синтезе указанного сигнала:

```
attribute FSM_EXTRACT : string;
attribute FSM_EXTRACT of <signal_name>: signal is "YES";
```

Каталог ***I/O***, входящий в состав раздела ***Synthesis***, имеет ту же структуру и содержание, что и одноименный раздел папки ***Attributes***. В каталоге ***Incremental Synthesis*** размещаются образцы применения директив, позволяющих использовать возможность инкрементного синтеза. Детализированная структура этого каталога представлена на рис. 528.

В состав каталога ***Incremental Synthesis*** входит единственный подкаталог ***True***, в котором находится образец записи директивы INCREMENTAL_SYNTHESIS, предназначенной для управления инкрементным синтезом.

Component (INCREMENTAL_SYNTHESIS) предоставляет шаблон конструкции, используемой для варианта применения атрибута INCREMENTAL_SYNTHESIS, разрешающего инкрементный синтез для указанного объекта описания:

```
attribute INCREMENTAL_SYNTHESIS : string;
attribute INCREMENTAL_SYNTHESIS of <entity_name>: entity
is "TRUE";
```

Модуль описания ***Info***, расположенный в разделе ***Synthesis*** папки ***Attributes***, содержит общую информацию об использовании специальных атрибутов в составе формируемого VHDL-описания проектируемого устройства. Здесь приведены краткие сведения о формате записи атрибутов и соответствующие примеры. Содержимое модуля описания ***Info*** выглядит следующим образом:

```
-- You can describe constraints with VHDL attributes in your VHDL
code. Before it can be
-- used, an attribute must be declared with the following syntax.
-- attribute AttributeName : string ;
-- Example
-- attribute RLOC : string ;
-- The attribute type defines the type of the attribute value.
-- An attribute can be declared in an entity or architecture. If declared
in the entity, it
-- is visible both in the entity and the architecture body. If the attribute
is declared in the
-- architecture, it cannot be used in the entity declaration. Once
declared a VHDL attribute
-- can be specified as follows:
-- attribute AttributeName of ObjectList : ObjectType is AttributeValue
;
-- Examples
-- attribute RLOC of u123 : label is R11C1.S0 ;
-- attribute BUFG of my_signal : signal is sr;
-- The object list is a comma separated list of identifiers. Accepted
object types are entity,
-- component, label, signal, variable and type.
```

Keep Hierarchy (KEEP_HIERARCHY) является образцом применения одноименного атрибута, который предоставляет возможность сохранения иерархической структуры исходного описания проекта в процессе синтеза. Предлагаемый вариант использования атрибута KEEP_HIERARCHY предназначен для сохранения определенного экземпляра

компонента в иерархической структуре синтезированного описания:

```
attribute KEEP_HIERARCHY : string;
attribute KEEP_HIERARCHY of <inst_name>: label is "TRUE";
```

После включения приведенной конструкции в состав модуля VHDL-описания следует заменить ***inst_name*** идентификатором экземпляра компонента, который не должен подвергаться преобразованию иерархической структуры в процессе синтеза.

Keep/Preserve a Signal (KEEP) содержит шаблон варианта записи атрибута KEEP, обеспечивающего сохранение указанной цепи (сигнала) в процессе синтеза исходного описания проектируемого устройства:

```
attribute KEEP : string;
attribute KEEP of <signal_name>: signal is "TRUE";
```

Каталог ***Multiplier Style*** раздела ***Synthesis*** объединяет образцы различных вариантов записи директивы MULT_STYLE, которая позволяет указать метод синтеза и последующей реализации макросов умножителей в кристаллах программируемой логики с архитектурой FPGA семейств Virtex-II, Virtex-II Pro, Virtex-II Pro X, Virtex-4 LX, Virtex-4 SX, Virtex-4 FX, Virtex-5 LX, Virtex-5 LXT, Virtex-5 SXT, Virtex-5 FXT, Virtex-6 LXT, Virtex-6 CXT, Virtex-6 SXT, Virtex-6 HXT, Spartan-3, Spartan-3E, Spartan-3L, Spartan-3A, Spartan-3AN, Spartan-3A DSP, Spartan-6 LX и Spartan-6 LXT. Развернутая структура этого каталога приведена на рис. 528.

В состав этого каталога входят семь подкаталогов — ***Auto***, ***Block***, ***CSD***, ***KCM***, ***Lut***, ***Pipe_block*** и ***Pipe_lut***. Каждый из этих подкаталогов соответствует одноименному способу синтеза и реализации макросов умножителей. В подкаталоге ***Auto*** находятся шаблоны вариантов записи директивы MULT_STYLE, предписывающих средствам синтеза автоматический выбор наилучшего метода последующей реализации для каждого обнаруженного макроса умножителя.

Auto/Entity (MULT_STYLE) включает в себя образец варианта применения атрибута MULT_STYLE, предоставляющего возможность выбора оптимального способа последующей реализации макросов умножителей при синтезе указанного объекта описания:

```
attribute: string;
attribute MULT_STYLE of <entity_name>: entity is "AUTO";
```

Auto/Signal (MULT_STYLE) демонстрирует шаблон варианта записи атрибута MULT_STYLE, обеспечивающего выбор наилучшего метода реализации для макросов умножителей при синтезе заданного сигнала:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <signal_name>: signal is "AUTO";
```

В подкаталоге ***Block*** расположены образцы вариантов использования атрибута MULT_STYLE для выполнения процесса синтеза с учетом последующего использования для выполнения операций умножения встроенных аппаратных блоков в кристаллах программируемой логики перечисленных выше семейств.

Block/Entity (MULT_STYLE) представляет собой шаблон варианта записи атрибута MULT_STYLE, предписывающего осуществление процесса синтеза указанного объекта описания с учетом реализации макросов умножителей на базе соответствующих встроенных аппаратных блоков ПЛИС:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <entity_name>: entity is "BLOCK";
```

Block/Signal (MULT_STYLE) содержит образец варианта применения атрибута MULT_STYLE для обеспечения в процессе синтеза заданного сигнала возможности последующей реализации операций умножения с помощью соответствующих встроенных аппаратных блоков кристаллов программируемой логики:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <signal_name>: signal is "BLOCK";
```

В подкаталоге ***CSD*** размещены шаблоны вариантов записи директивы MULT_STYLE, используемых для осуществления синтеза макросов умножителей в соответствии с методом Canonical Signed Digit. Этот вариант применяется при синтезе умножителей, в которых один из входных операндов представлен в виде константы со знаком.

CSD/Entity (MULT_STYLE) является образцом варианта применения атрибута MULT_STYLE для выбора метода Canonical Signed Digit при выполнении синтеза указанного объекта описания:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <entity_name>: entity is "CSD";
```

CSD/Signal (MULT_STYLE) включает в себя шаблон варианта записи атрибута MULT_STYLE, предписывающего использование метода Canonical Signed Digit при осуществлении синтеза заданного сигнала:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <signal_name>: signal is "CSD";
```

В подкаталоге ***KCM*** расположены образцы вариантов применения директивы MULT_STYLE для выбора метода Constant Coefficient Multiplier при синтезе макросов умножителей. Эти варианты используются при синтезе макросов, выполняющих операцию умножения на постоянный коэффициент.

KCM/Entity (MULT_STYLE) предоставляет шаблон варианта записи атрибута MULT_STYLE, обеспечивающего применение метода Constant Coefficient Multiplier при проведении синтеза соответствующего объекта описания:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <entity_name>: entity is "KCM";
```

KCM/Signal (MULT_STYLE) демонстрирует образец варианта использования атрибута MULT_STYLE для выбора метода Constant Coefficient Multiplier в процессе синтеза указанного сигнала:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <signal_name>: signal is "KCM";
```

В подкаталоге ***Lut*** сосредоточены шаблоны вариантов записи директивы MULT_STYLE, предназначенных для осуществления синтеза макросов умножителей с учетом их последующей реализации на базе ресурсов таблиц преобразования LUT кристаллов программируемой логики с архитектурой FPGA.

Lut/Entity (MULT_STYLE) содержит шаблон варианта записи атрибута MULT_STYLE, предписывающего выполнение процесса синтеза заданного объекта описания с учетом реализации макросов умножителей на основе таблиц преобразования LUT:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <entity_name>: entity is "LUT";
```

Lut/Signal (MULT_STYLE) представляет собой образец варианта использования атрибута MULT_STYLE для осуществления процесса синтеза указанного сигнала с учетом последующей реализации операций умножения на базе таблиц преобразования LUT:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <signal_name>: signal is "LUT";
```

Подкаталог ***Pipe_block*** объединяет шаблоны вариантов записи директивы MULT_STYLE, применяемых для выполнения процессов синтеза макросов умножителей и последующей реализации на основе встроенных аппаратных блоков умножения в сочетании с конвейерными регистрами.

Pipe_block/Entity (MULT_STYLE) является образцом варианта применения атрибута MULT_STYLE, позволяющего учитывать в процессе синтеза определенного объекта описания реализацию макросов умножителей на основе соответствующих аппаратных блоков ПЛИС в сочетании с конвейерными регистрами:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <entity_name>: entity is "PIPE_BLOCK";
```

Pipe_block/Signal (MULT_STYLE) включает в себя шаблон варианта записи атрибута MULT_STYLE, предписывающего выполне-

ние синтеза заданного сигнала с учетом последующего использования встроенных аппаратных умножителей и дополнительных регистров при реализации операций умножения:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <signal_name>: signal is "PIPE_BLOCK";
```

В подкаталоге ***Pipe_lut*** сгруппированы образцы вариантов применения директивы MULT_STYLE, обеспечивающих осуществление процесса синтеза макросов умножителей для последующей реализации на основе таблиц преобразования LUT с привлечением дополнительных регистров.

Pipe_lut/Entity (MULT_STYLE) предоставляет шаблон варианта записи атрибута MULT_STYLE, предназначенного для выбора алгоритмов синтеза указанного объекта описания, в результате осуществления которых для реализации операций умножения задействуются таблицы преобразования LUT и дополнительные конвейерные регистры:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <entity_name>: entity is "PIPE_LUT";
```

Pipe_lut/Signal (MULT_STYLE) демонстрирует образец варианта применения атрибута MULT_STYLE, позволяющего учитывать при синтезе указанного сигнала реализацию умножителей на базе таблиц преобразования LUT и конвейерных регистров:

```
attribute MULT_STYLE : string;
attribute MULT_STYLE of <signal_name>: signal is "PIPE_LUT";
```

Примечание. *Полный список литературы смотрите в предыдущих частях статьи.*

Продолжение следует