Школа схемотехнического проектирования устройств обработки сигналов

Занятие 10.
Языки описания аппаратуры: синтаксис и особенности применения

Владимир Стешенко

steshenk@sm.bmstu.ru

Исходный текст описания на VHDL состоит из последовательностей операторов, записанных с учетом следующих правил:

- каждый оператор — это последовательность слов, содержащих буквы английского алфавита, цифры и знаки пунктуации;
- слова разделяются произвольным количеством пробелов, табуляций и переводов строки;
- операторы разделяются символами «;»;
- в некоторых операторах могут встречаться списки объектов, разделяемые символами «,» или «;».

Комментарии могут быть включены в текст программы с помощью двух подряд идущих символов «–». После появления этих символов весь текст до конца строки считается комментарием.

Для указания системы счисления для констант могут быть применены спецификаторы: B — двоичная система счисления (например, B»0011»), O — восьмеричная система счисления (например, O»3760») и H — шестнадцатеричная система счисления (например, H»F6A0»).

Одним из наиболее важных понятий VHDL являются объекты.

Объекты являются контейнерами для хранения различных значений в рамках модели. Идентификаторы объектов содержат буквы, цифры и знаки подчеркивания. Они должны начинаться с буквы и не должны заканчиваться знаком подчеркивания. Знаки подчеркивания не могут идти подряд. Прописные и строчные буквы в VHDL не различаются. Все объекты должны быть явно объявлены перед использованием, за исключением переменной цикла в операторе *for*, которая объявляется по умолчанию.

Каждый объект характеризуется типом и классом. Тип показывает, какого рода данные может содержать объект. Типы разделяются на предопределенные в VHDL и определяемые пользователем. Класс показывает, что можно сделать с данными, содержащимися в объекте. В VHDL определены следующие классы объектов:

- **Constant** — константы. Значение определяется при ее объявлении и не может быть изменено. Константы могут иметь любой из поддерживаемых типов данных.
- **Variable** — переменные. Значение, хранимое в переменной, может изменяться везде, где встречается присваивание этой переменной. Переменные также могут иметь любой из поддерживаемых типов данных.
- **Signal** — сигналы. Они представляют значения, передаваемые по проводам и определяемые присвоением сигналов (отличным от присвоения переменных). Сигналы могут иметь ограниченный набор типов (обычно *bit, bit_vector, std_logic, std_logic_vector, integer* и др., в зависимости от среды разработки).

Повторное использование присваивания сигналов в наборе параллельных операторов не допускается. В наборе последовательных операторов такое присваивание допустимо и даст значение сигнала, соответствующее последнему по порядку присваиванию.

Атрибуты (свойства) определяют характеристики объектов, к которым они относятся. Стандарт VHDL предусматривает как предопределенные, так и определяемые пользователем атрибуты, однако современные инструментальные средства в большинстве своем поддерживают только предопределенные атрибуты. Для обращения к атрибутам объекта используется символ «'» (например, *A1'left*).

В VHDL определены следующие атрибуты:

- 'left — левая граница диапазона индексов массива;
- 'right — правая граница диапазона индексов массива;
- 'low — нижняя граница диапазона индексов массива;
- 'high — верхняя граница диапазона индексов массива;
- 'range — диапазон индексов массива;
- 'reverse_range — обращенный диапазон индексов массива;
- 'length — ширина диапазона индексов массива.

Объявление компонента определяет интерфейс к модели на VHDL (*entity* и *architecture*), описанной в другом файле. Обычно объявление компонента совпадает с соответствующим объявлением entity. Они могут различаться только значениями по умолчанию. Эти значения используются, когда какой-либо

из отводов компонента остается неприсоединенным (ключевое слово *open*) при установке компонента в схему.

Оператор объявления компонента может находиться внутри объявления *architecture* или в заголовке пакета (*package*). Соответствующие компоненту объявления *entity* и *architecture* не обязательно должны существовать в момент анализа схемы. В момент моделирования или синтеза должны существовать объявления *entity* и *architecture* для компонентов, которые не только объявлены, но и установлены в схему. Это позволяет, например, конструктору задать объявления библиотечных элементов, а реальное их описание (объявления *entity* и *architecture*) задавать по мере использования этих элементов в конструкции.

Объявление компонента записывается следующим образом:

```
Component name
[ port( port_list ); ]
end component;
```

Выражения в языке VHDL могут содержать следующие операторы: преобразование типа *and, or, nand, nor, xor, =, /=, <, <=, >, >=, +, -, &, *, /, mod, rem, abs, not.*

В зависимости от избранной САПР при синтезе может поддерживаться подмножество приведенных выше операторов. С помощью операторов описывается алгоритм, определяющий функционирование схемы. Они могут находиться в теле функции, процедуры или процесса. Порядок вычисления выражений определяется приоритетом операторов.

Оператор *Wait until condition* приостанавливает выполнение процесса, содержащего данный оператор до момента выполнения условия.

Оператор присваивания сигнала *Signal <= expression* устанавливает его значение равным выражению справа.

Оператор присваивания *Variable:= expression* устанавливает значение переменной равным выражению справа.

Оператор вызова процедуры *Procedure_name (parameter {, parameter })* состоит из имени процедуры и списка фактических параметров.

Оператор *if* используется для ветвления алгоритма по различным условиям.

```
If condition then
Sequence_of_statements
{ Elsif condition then
Sequence_of_statements }
[ else
sequence_of_statements ]
end if ;
```

Оператор *case*, подобно оператору *if*, задает ветвление алгоритма. Значения в списках разделяются символом «|». Когда значение выражения встречается в одном из списков значений, выполняется соответствующая последовательность операторов. Если значение выражения не присутствует ни в одном из списков, то выполняется список операторов, соответствующий ветви *when others*.

Оператор цикла позволяет многократно выполнить последовательность операторов. Диапазон значений задается в виде *value1 to value2* или *value1 downto value2*. Переменная цикла последовательно принимает значения из заданного диапазона. Количество итераций равно количеству значений в диапазоне.

Оператор *Return expression* возвращает значение из функции.

Null — пустой оператор, не выполняет никаких действий.

Полное VHDL-описание объекта состоит как минимум из двух отдельных описаний: описание интерфейса объекта и описание тела объекта (описание архитектуры).

Интерфейс описывается в объявлении объекта *entity declaration* и определяет входы и выходы объекта, его входные и выходные порты *ports* и параметры настройки *generic*. Параметры настройки отражают тот факт, что некоторые объекты могут иметь управляющие входы, с помощью которых может производиться настройка экземпляров объектов, в частности задаваться время задержки.

Например, у объекта Q1 три входных порта X1, X2, X3 и два выхода У1, У2. Описание его интерфейса на VHDL имеет вид:

```
Entity Q1 is
Port (X1, X2, X3: in real; Y1, Y2: out real);
End Q1.
```

Порты объекта характеризуются направлением потока информации. Они могут быть входными (in) и выходными (out), двунаправленными (inout) и двунаправленными буферными (buffer), а также связными (linkage).

Порты имеют тип, характеризующий значения поступающих на них сигналов:

- целый (integer);
- вещественный (real);
- битовый (bit);
- символьный (character).

Тело объекта специфицирует его структуру или поведение. Его описание по терминологии VHDL содержится в описании его архитектуры *architecture*.

VHDL позволяет отождествлять с одним и тем же интерфейсом несколько архитектур. Это связано с тем, что в процессе проектирования происходит проработка архитектуры объекта: переход от структурной схемы к электрической принципиальной, от поведенческого к структурному описанию.

Средства VHDL для отображения структур цифровых систем базируются на представлении о том, что описываемый объект *entity* представляет собой структуру из компонентов *component*, соединяемых друг с другом линиями связи. Каждый компонент, в свою очередь, является объектом и может состоять из компонентов низшего уровня (иерархия объектов). Взаимодействуют объекты путем передачи сигналов *signal* по линиям связи. Линии связи подключаются к входным и выходным портам компонентов. В VHDL сигналы отождествляются с линиями связи.

Имена сигналов и имена линий связи совпадают (они отождествляются). Для сигналов (линий), связывающих компоненты друг с другом, необходимо указывать индивидуальные имена.

Описание структуры объекта строится как описание связей конкретных компонентов, каждый из которых имеет имя, тип и карты портов. Карта портов *port map* определяет соответствие портам компонентов, поступающим на них в виде сигналов. Можно интерпретировать карту портов как разъем, на который приходят сигналы и в который вставляется объект компонента.

Средства VHDL для отображения поведения описываемых архитектур строятся на представлении их как совокупности параллельно взаимодействующих процессов. Понятие процесса process относится к базовым понятиям языка VHDL.

Архитектура включает в себя описание одного или нескольких параллельных процессов. Описание процесса состоит из последовательности операторов, отображающих действия по переработке информации. Все операторы внутри процесса выполняются последовательно. Процесс может находиться в одном из двух состояний — либо пассивном, когда процесс ожидает прихода сигналов запуска или наступления соответствующего момента времени, либо активном — когда процесс исполняется.

Процессы взаимодействуют путем обмена сигналами.

В общем случае в поведенческом описании состав процессов не обязательно соответствует составу компонентов, как в структурном описании.

Поведение VHDL-объектов воспроизводится на компьютере. При этом приходится учитывать особенности воспроизведения параллельных процессов на однопроцессорной ЭВМ. Особая роль в синхронизации процессов отводится механизму событийного воспроизведения модельного времени *now*.

Когда процесс вырабатывает новое значение сигнала перед его посылкой на линию связи, говорят, что он вырабатывает будущее сообщение *transaction*. С каждой линией связи (сигналом) может быть связано множество будущих сообщений. Множество сообщений для сигнала называется его драйвером *driver*. Таким образом, драйвер сигнала — это множество пар типа «время — значение» (множество планируемых событий).

VHDL реализует механизм воспроизведения модельного времени, состоящий из циклов. На первой стадии цикла вырабатываются новые значения сигналов. На второй стадии процессы реагируют на изменения сигналов и переходят в активную фазу. Эта стадия завершается, когда все процессы снова переходят в состояние ожидания. После этого модельное время становится равным времени ближайшего запланированного события, и все повторяется.

Особый случай представляет ситуация, когда в процессах отсутствуют операторы задержки. Для этого в VHDL предусмотрен механизм так называемой дельта-задержки. В случае дельта-задержек новый цикл моделирования не связан с увеличением модельного времени.

Другая способность VHDL-процессов связана с так называемыми разрешенными (*resolved*) сигналами. Если несколько процессов изменяют один и тот же сигнал, то есть сигнал имеет несколько драйверов, в описании объектов может указываться функция разре-

шения. Эта функция объединяет значения из разных драйверов и вырабатывает одно. Это позволяет, например, работать нескольким элементам на общую шину.

В языке VHDL для наиболее часто используемых видов процессов — процессов межрегистровых передач — введена компактная форма записи.

Для иллюстрации возможностей VHDL рассмотрим пример проектирования простой комбинационной схемы. Назовем ее объект F. Объект проекта F имеет два входа A1 и A2 и два выхода B1 и B2.

Объявление объекта проекта F:

```
Entity F is
Port (A1, A2: in BIT; B1, B2: out BIT)
```

Сигналы принимают значения 1 или 0 в соответствии с таблицей истинности (таблица 3).

Таблица 3

Входы		Выходы	
A1	A2	B1	B2
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Вариант описания архитектуры BEHAVIOR объекта F использует условный оператор *if* языка VHDL и учитывает, что только при обоих входах A1 и A2, равных 1, выходы B1 = 1 и B2 = 0. В остальных случаях наоборот, B1 = 0 и B2 = 1.

```
Architecture BEHAVIOR of F is
Begin
Process
Begin
Wait on (A1, A2)
If (A1='1') and (A2='1')
Then B1<='1'; B2<='0';
End if;
End process;
End;
```

В каждом процессе может быть только один оператор *wait on*. Второй вариант поведенческого описания архитектуры объекта F, назовем его BEHAVIOR_F, использует выбор *case* языка VHDL и учитывает следующее свойство функции F: для первых трех строк ее значение не меняется. В заголовке процесса указан список чувствительности процесса *process (A1, A2)*. Это указание эквивалентно оператору *wait on (A1, A2)* в начале описания процесса.

```
Architecture BEHAVIOR_F of F is
Begin
Process (A1,A2);
Begin
--&-операция
case (A1& A2) is
--первые три строки таблицы
when «00»/ «01»/ «10»=> B1<='0'; B2<='1'
--последняя строка таблицы
when «11» => B1<='1'; B2<='0'
end case
end process
end BEHAVIOR_F;
```

В процессе проектирования объекта F могут быть предложены различные варианты его функциональных схем. Описание архитектуры объекта F может быть таким:

```
Architecture F_A of F is
Begin
--каждому вентилю сопоставлен оператор назначения сигнала
B1<= A1 and A2;
B2<= not (A1 and A2);
End;
```

Здесь каждому элементу сопоставлен процесс, отображающий последовательность преобразования входной информации и передачи ее на выход. Процесс представлен в форме оператора параллельного назначения сигнала. Операторы назначения сигнала (<=) срабатывают параллельно при изменении хотя бы одного из сигналов в своих правых частях.

Другой вариант описания архитектуры F_B. Здесь вентили включены последовательно.

```
Architecture F_B of F is
Signal X: bit
Begin
B2<= not (X);
X <= A1 and A2;
B1 <= X;
End;
```

Промежуточный сигнал X введен в описание архитектуры F_B объекта F потому, что в описании интерфейса объекта F порт B1 объявлен выходным, то есть с него нельзя считывать сигнал, и запись B2<= not(B1) была бы некорректной.

Сигнал B2 вырабатывается только после изменения сигнала X. Оператор B2<= not(X) сработает только тогда, когда изменится сигнал X, то есть после оператора X<= A1 and A2, так как он реагирует только на изменение сигнала в своей правой части. С учетом задержек E1 = 10 нс и E2 = 5 нс описание архитектуры будет иметь вид:

```
Architecture F_B_TIME of F is
Signal X: bit
Begin
--задержка на B1- 10 нс.
--задержка на B2- 5 нс.
B1<=X;
B2<= not (X) after 5 ns;
X<= A1 and A2 after 10 ns;
End;
```

Через 10 нс после изменения одного из входных сигналов (A1 или A2) может измениться выходной сигнал B1, и с задержкой 5 нс после него изменится B2.

Описание архитектуры представляет собой структуру объекта как композицию соединенных между собой компонентов, обменивающихся сигналами. Функции, реализуемые компонентами в явном виде, в отличие от предыдущих примеров в структурном описании не указываются. Структурное описание включает описание интерфейсов компонентов, из которых состоит схема, и их связей. Полные (интерфейс + архитектура) описания объектов-компонентов должны быть ранее помещены в проектируемую библиотеку, подключенную к структурному описанию архитектуры.

Одним из средств повышения компактности описаний цифровых устройств является использование векторных представлений сигналов и операций над ними. Например, пусть некоторый объект FV выполняет ту же функцию, что и объект F, но над 20-разрядными двоичными векторами AV1 и AV2.

Его описание определяет порты как двоичные векторы:

```
Entity FV is
Port (AV1, AV2: in bit_vector (1 to 20);
BV1, BV2: out bit_vector (1 to 20));
End FV1;
```

Поведенческое описание архитектуры FV в потоковой форме использует операции над битовыми векторами.

```
Architecture BECHAV_FV of FV is
Begin
BV2<= not (AV1 and AV2);
BV1<= AV1 and AV2;
End BECHAV_FV;
```

Структурное описание архитектуры FV для варианта реализации объекта FV как совокупности объектов F, представленное ниже, выполнено с использованием оператора генерации конкретизации. Это позволяет повысить компактность описаний регулярных фрагментов схем.

```
Architecture STRUCT_FV of FV is
Component F port (X1, X2: in bit; Y1, Y2: out bit);
End component;
Begin
--первая компонента конкретизирована обычным способом с использованием позиционного соответствия сигналов портам
K1: F port map (AV1 (1), AV2 (1), BV1 (1), BV2 (1));
--вторая компонента конкретизирована с использованием ключевого способа указания соответствия сигналов ее портам
K2: F port map (AV1 ( 2)=>X1, BV1 ( 2)=>Y1, AV2 (2)=>X2, BV2 (2) =>Y2 );
--компоненты K3 — K20 конкретизированы с использованием оператора генерации, позволяющего компактно описывать регулярные фрагменты схем
for I in 3 to 20 generate
K( I ): F port map ( AV1 ( 1 ), AV2 (1), BV1 (1), BV2 (1) );
End generate;
End STRUCT_FV;
```

Объект с задержкой можно представить состоящим из двух — идеального элемента и элемента задержки. В языке VHDL встроены две модели задержек — инерциальная и транспортная. Инерциальная модель предполагает, что элемент не реагирует на сигналы, длительность которых меньше порога, равного времени задержки элемента. Транспортная модель лишена этого ограничения.

Инерциальная модель по умолчанию встроена в оператор назначения сигнала языка VHDL. Например, оператор назначения *Y<=X1 and X2 after 10 ns*; описывает работу вентиля 2И и соответствует инерциальной модели. Указание на использование транспортной модели обеспечивается ключевым словом *transport* в правой части оператора назначения. Например, оператор *YT<=transport X1 and X2 after 10 ns*; отображает транспортную модель задержки вентиля.

Задержка может быть задана не константой, а выражением, значение которого может конкретизироваться для каждого экземпляра объекта, используемого как компонента. Для этого ее следует задать как параметр настройки в описании интерфейса объекта.

Описания систем могут содержать информацию о запрещенных ситуациях, например о недопустимых комбинациях сигналов на входах объектов, рекомендуемых длительностях или частотах импульсов и т. п. Например,

в вентиле 2И возникает риск сбоя в ситуациях, когда фронт одного сигнала перекрывает срез другого.

Средством отображения информации о запрещенных ситуациях в языке VHDL является оператор утверждения (оператор контроля, оператор аномалии) *assert*. В нем, помимо контролируемого условия, которое не должно быть нарушено, то есть должно быть истинным, записывается сообщение *report* о нарушении и уровень серьезности ошибки *severity*.

Более полное представление о предопределенных атрибутах сигналов можно получить из табл. 4. Помимо предопределенных, пользователь может вводить дополнительные атрибуты для сигналов и других типов данных.

Таблица 4. Предопределенные атрибуты сигналов

Пример	Тип результата	Пояснения
S'QUIET(T)	Boolean	TRUE, если сигнал S пассивен на интервале T
S'TRANSACTION	Bit	Инвертируется S каждый раз, когда S активен (изменяется)
S'STABLE(T)	Boolean	TRUE, если не было событий за интервал T
S'DELAYED(T)	Signal	Предыдущее значение S в момент NOW-T
S'ACTIVE	Boolean	TRUE, если сигнал активен
S'LAST_ACTIVE	Time	Время, когда сигнал последний раз был активен
S'EVENT	Boolean	TRUE, если происходит событие в S
S'LAST_VALUE	Signal	Значение сигнала перед последним событием в нем
S'LAST_EVENT	Time	Время последнего события в S

Описание пакета VHDL задается ключевым словом *package* и применяется с целью собирать часто используемые элементы конструкции для использования глобально в других проектах. Пакет можно рассматривать как общую область хранения описаний типов, констант и глобальных подпрограмм. Объекты, определенные в пределах пакета, можно использовать в любом другом проекте на VHDL и можно скомпилировать в библиотеки для дальнейшего повторного использования.

Пакет может состоять из двух основных частей: описания пакета и дополнительного тела пакета. Описание пакета может содержать следующие элементы:

- объявления типов и подтипов;
- объявления констант;
- глобальные описания сигналов;
- объявления процедур и функций;
- спецификация атрибутов;
- объявления файлов;
- объявления компонентов;
- объявления псевдонимов;
- операторы включения.

Пункты, появляющиеся в пределах описания пакета, могут стать видимыми в других проектах с помощью оператора включения.

Если пакет содержит описания подпрограмм (функций или процедур) или определяет одну или более задерживаемых констант (константы, чья величина не задана), то в дополнение к описанию необходимо тело пакета. Тело пакета (которое определяется с использованием комбинации ключевых слов «*package body*»), должно иметь то же имя, что и соответствующее описание пакета, но может располагаться в любом месте проекта (не обязательно сразу после описания пакета).

Отношение между описанием и телом пакета отчасти напоминает отношение между описанием и реализацией элемента (тем не менее может быть только одно тело пакета для каждого описания пакета). В то время как описание пакета обеспечивает информацию, необходимую для использования элементов, определенных в пределах этого пакета (список параметров для глобальной процедуры, или имя определенного типа или подтипа), фактическое поведение таких объектов, как процедуры и функции, должно определяться в пределах тела пакета.

Приведенные выше описания объекта F базировались на стандартных средствах языка VHD — сигналы представлялись в алфавите '1', '0', логические операции И, ИЛИ, НЕ также определялись в этом алфавите. Во многих случаях приходиться описывать поведение объектов в других алфавитах. Например, в реальных схемах сигнал кроме значений '1' и '0' может принимать значение высокого импеданса 'Z' (на выходе буферных элементов) и неопределенное значение 'X', например, отражая неизвестное начальное состояние триггеров. Переход к другим алфавитам осуществляется в VHDL с помощью пакетов.

Ниже приводится пример пакета P4 для описания объектов в четырехзначном алфавите представления сигналов ('X', '0', '1', 'Z'), X — значение не определено, Z — высокий импеданс. В этом пакете приводится тип KONTAKT для представления сигналов в четырехзначном алфавите и определяются функции NOT и AND над ними.

В ТТЛ логике высокий импеданс на входе воспринимается как 1, что учитывается в таблице функции AND.

Ниже следует объявление пакета P4:

```
Package P4 is
--перечислимый тип
type KONTAKT is ('X', '0', '1', 'Z' );
function «NOT» (X: in KONTAKT) return KONTAKT;
function «AND» (X1, X2: in KONTAKT) return KONTAKT;
end P4;
```

Ниже следует объявление тела пакета P4:

```
Package body P4 is
Function «NOT» (A: in KONTAKT) return KONTAKT is
Begin
If A='X' then return 'X'
Else if A='1' then return '0'
Else if A='0' then return '1'
Else return 'Z'
End if;
End «NOT»;
Function «AND» ( A1, A2: in KONTAKT ) return KONTAKT is
Begin
If (A1='0' ) or (A2='X' ) then return to '0'
Else if (A1= 'X' ) or (A2='0' ) or (A2= 'X') and (A1='0' ) then return 'X'
Else return to'1'
End if;
End «AND»;
End P4;
```

Ниже приведен пример использования пакета P4 при описании объекта F_P4. Этот объект отличается от F, так как у него другой интерфейс.

```
--подключается ( use) пакет P4, все его функции (ALL)
use P4.ALL;
entity F_P4 is
port (A1, A2: in KONTAKT; B1, B2: out KONTAKT )
end F_P4;
```

Описание архитектуры F_P4_a

```
Architecture F_P4_a of F is
Begin
B2<= not ( A1 and A2 );
B1<= A1 and A2;
End F_P4_a;
```

Из этого примера видно, что в ряде случаев изменение алфавита моделирования не требует внесения изменений в описания объектов. Например, переход к семизначному алфавиту ('0', '1', 'X', 'Z', 'F', 'S', 'R'), где тип KONTAKT имеет дополнительные значения (F — фронт, S — срез, R — риск сбоя), потребует только создания нового пакета и подключения его к объявлению объекта F_P4. Изменение в других частях описаний объекта проекта не потребуется.

В языке VHDL предусмотрен механизм разрешения конфликтов, возможных в ситуациях, когда сигнал имеет несколько драйверов. Функция разрешения обычно описывается в пакете, а ее имя указывается при описании соответствующего сигнала. Например, тело функции разрешения WIRED_OR (монтажное ИЛИ) имеет следующий вид:

```
Function WIRED_OR (INPUTS: bit_vector) return bit is
Begin
For I in INPUTS'RANGE loop
If INPUTS( I ) = '1' then
Return '1' ;
End if;
End loop;
Return '0';
End;
```

Драйверы сигнала INPUTS неявно рассматриваются как массив, границы которого определяются атрибутом 'RANGE. Функция сканирует драйверы сигнала и, если хоть один из них равен '1', возвращает значение '1', иначе '0'.

Функция разрешения SHIN для шины на элементах с тремя состояниями выходами может быть такой:

```
Type A3 is ('0', '1', 'Z');
Type VA3 is array ( integer range <> of A3 );
Function SHIN (signal X: VA3 ) return A3 is
Variable VIXOD: A3:= 'Z';
Begin
For I in X'RANGE loop
I f X( I) /= 'Z' then
VIXOD:= X ( I);
Exit;
End if;
End loop;
Return VIXOD;
End SHIN;
```

Предполагается, что может быть включен (то есть не равен 'Z') только один из драйверов входных сигналов.

На этом завершим обзор особенностей языка VHDL, заинтересованный читатель может найти информацию как в литературе, так и в Интернете (**w ww. vhdl.o rg**).