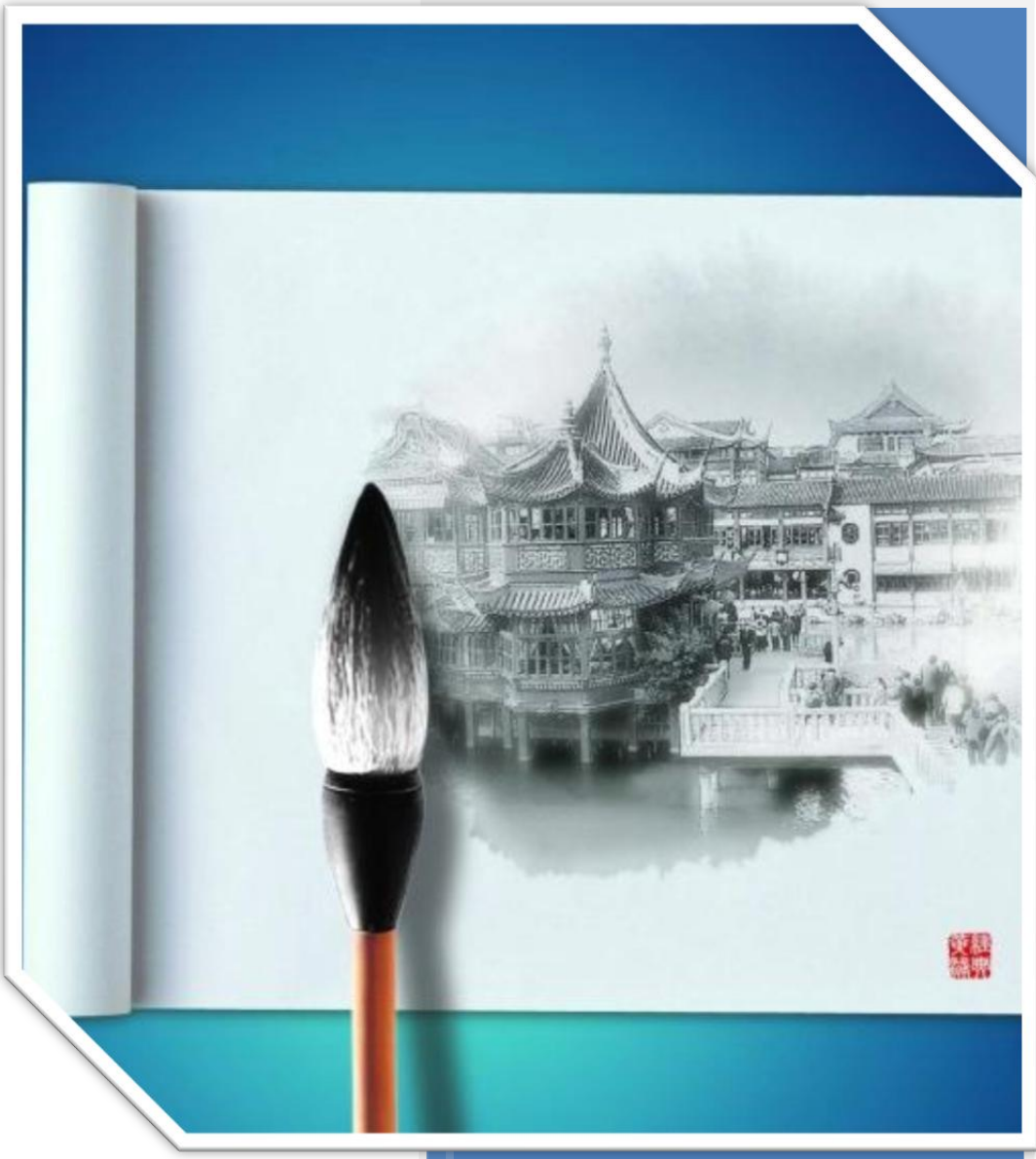
FLQ100ITOI-V1.0

HDL 基础语法篇 —— VHDL



zrtech

WWW.ZR-TECH.COM

VHDL硬件描述语言

1.1 VHDL概述

1.1.1 VHDL的特点

VHDL语言作为一种标准的硬件描述语言，具有结构严谨、描述能力强的特点，由于VHDL语言来源于C、Fortran等计算机高级语言，在VHDL语言中保留了部分高级语言的原语句，如if语句、子程序和函数等，便于阅读和应用。具体特点如下：

1. 支持从系统级到门级电路的描述，既支持自底向上（bottom-up）的设计也支持从顶向下（top-down）的设计，同时也支持结构、行为和数据流三种形式的混合描述。

2. VHDL的设计单元的基本组成部分是实体（entity）和结构体（architecture），实体包含设计系统单元的输入和输出端口信息，结构体描述设计单元的组成和行为，便于各模块之间数据传送。利用单元（componet）、块（block）、过程（procure）和函数（function）等语句，用结构化层次化的描述方法，使复杂电路的设计更加简便。采用包的概念，便于标准设计文档资料的保存和广泛使用。

3. VHDL语言有常数、信号和变量三种数据对象，每一个数据对象都要指定数据类型，VHDL的数据类型丰富，有数值数据类型和逻辑数据类型，有位型和位向量型。既支持预定义的数据类型，又支持自定义的数据类型，其定义的数据类型具有明确的物理意义，VHDL是强类型语言。

4. 数字系统有组合电路和时序电路，时序电路又分为同步和异步，电路的动作行为有并行和串行动作，VHDL语言常用语句分为并行语句和顺序语句，完全能够描述复杂的电路结构和行为状态。

1.1.2 VHDL语言的基本结构

VHDL语言是数字电路的硬件描述语言，在语句结构上吸取了Fortran和C等计算机高级语言的语句，如IF语句、循环语句、函数和子程序等，只要具备高级语言的编程技能和数字逻辑电路的设计基础，就可以在较短的时间内学会VHDL语言。但是VHDL毕竟是一种描述数字电路的工业标准语言，该种语言的标识符号、数据类型、数据对象以及描述各种电路的语句形式和程序结构等方面具有特殊的规定，如果一开始就介绍它的语法规定，会使初学者感到枯燥无味，不得要领。较好的办法是选取几个具有代表性的VHDL程序实例，先介绍整体的程序结构，再逐步介绍程序中的语法概念。

一个VHDL语言的设计程序描述的是一个电路单元，这个电路单元可以是一个门电路，或者是一个计数器，也可以是一个CPU。**一般情况下，一个完整的VHDL语言程序至少要包含程序包、实体和结构体三个部分。实体给出电路单元的外部输入输出接口信号和引脚信息，结构体给出了电路单元的内部结构和信号的行为特点，程序包定义在设计结构体和实体中将用到的常数、数据类型、子程序和设计好的电路单元等。**

例1-1-1用VHDL语言描述一位全加器。一位全加器的输入信号是A，B，Ci，输出信号是S和Co。全加器的真值表如表1-1-1所列。

表1-1-1

全加器的真值表输入信号			输出信号	
A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0

0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

一位全加器的逻辑表达式是：

$S=A \oplus B \oplus Ci$

$Co=AB+ACi+BCi$

全加器的VHDL程序的文件名称是fulladder.VHD，其中VHD是VHDL程序的文件扩展名，程序如下：

```
LIBRARYIEEE;                                    --IEEE标准库

USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY fulladder IS                             -- fulladder是实体名称
PORT(
        A, B, Ci    : IN    STD_LOGIC;          --定义输入/输出信号
        Co, S       : OUT   STD_LOGIC
    );
END fulladder;
ARCHITECTURE addstr OF fulladder IS     --addstr是结构体名
BEGIN
S <= A XOR B XOR Ci;
Co <= (A AND B) OR (A AND Ci) OR (B AND Ci);
END addstr;
```

从这个例子中可以看出，一段完整的VHDL代码主要由以下几部分组成：

第一部分是程序包，程序包是用VHDL语言编写的共享文件，定义在设计结构体和实体中将用到的常数、数据类型、子程序和设计好的电路单元等，放在文件目录名称为IEEE的程序包库中。

第二部分是程序的实体，定义电路单元的输入/输出引脚信号。程序的实体名称fulladder是任意取的，但是必须与VHDL程序的文件名称相同。实体的标识符是ENTITY，实体以ENTITY开头，以END结束。其中，定义A、B、Ci是输入信号引脚，定义Co和S是输出信号引脚。

第三部分是程序的结构体，具体描述电路的内部结构和逻辑功能。结构体有三种描述方式，分别是行为（BEHAVIOR）描述、数据流（DATAFLOW）描述方式和结构(STRUCTURE)描述方式，其中数据流（DATAFLOW）描述方式又称为寄存器（RTL）描述方式，例中结构体的描述方式属于数据流描述方式。结构体以标识符 ARCHITECTURE 开头，以 END 结尾。结构体的名称 addstr 是任意取的。

小提示：
VHDL 每条语句是以分号“；”作为结束符的，并且 VHDL 对空格是不敏感的，所以符合之间空格的数目是可以自己设定的。可以按自己的习惯任意添加，增强代码可读性。

1.1.3 VHDL语言的实体（ENTITY）说明语句

实体是VHDL程序设计中最基本的组成部分，在实体中定义了该设计芯片中所需要的输入/输出信号引脚。端口信号名称表示芯片的输入/输出信号的引脚名，这种端口信号通常被称为外部信号，信号的输入/输出状态被称为端口模式，在实体中还定义信号的数据类型。实体说明语句的格式为：

```
ENTITY 实体名称 IS
GENERIC（
常数名称1：类型 [:=缺省值];
常数名称2：类型 [:=缺省值];
…
常数名称N：类型 [:=缺省值];
）;
PORT（
端口信号名称1：输入/输出状态数据类型;
端口信号名称2：输入/输出状态数据类型;
…
端口信号名称N：输入/输出状态数据类型
）;
END 实体名称;
```

小提示：
VHDL 语言具有 87 标准与 93 标准两种格式，以上为 VHDL 的 87 标准，对于 93 标准 要使用 END ENTITY 实体名称；结束实体。注意为了保证代码的可综合性与通用性，最好采用 87 标准的 VHDL 格式，有些 EDA 工具不一定支持 93 标准的 VHDL 语言格式。（Quartus II 支持 VHDL93、87 标准）

类属GENERIC常用来定义实体端口大小，数据宽度，元件例化数目等。一般在简单的设计中不常用。

例1-1-2一个同步十六进制加法计数器，带有计数控制、异步清零、和进位输出等功能。电路有三个输入端和五个输出端，分别是时钟脉冲输入端CLK，计数器状态控制端EN，异步清零控制端Rd，四位计数输出端Q0，Q1，Q2，Q3和一个进位输出端Co。当计数器输出0000～1110时，Co=0，只有当计数器输出1111时，Co=1。电路的功能表如表1-1-2所示。

表1-1-2

同步十六进制加法计数器的功能表控制端			工作状态
CLK	EN	Rd	
×	×	0	异步清

			零
上升沿	1	1	计数
×	0	1	保持

该设计的实体部分如下：

```
ENTITY cntm16 IS
PORT(
        EN : IN STD_LOGIC;
        Rd : IN STD_LOGIC;
        CLK : IN STD_LOGIC;
        Co : OUT STD_LOGIC;
        Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0)
      );
END cntm16;
```

1. 实体名称表示所设计电路的电路名称，必须与VHDL文件名相同，实体名称是“cntm16”，所存的VHDL文件名必须是 “cntm16.VHD”。

2. 端口信号名称表示芯片的输入/输出信号的引脚名，这种端口信号通常被称为外部信号，端口信号名称可以表示一个信号，也可以表示一组信号（BUS），由数据类型定义，如EN，Rd，CLK，Co分别表示计数允许信号，异步清零信号，时钟输入信号和进位输出信号，Q是一组输出信号，用来表示四位同步二进制计数器的四位计数输出信号。

3. 端口信号输入/输出状态有以下几种状态：

IN 信号进入电路单元。

OUT 信号从电路单元输出。

INOUT 信号是双向的，既可以进入电路单元也可以从电路单元输出。

BUFFER 信号从电路单元输出，同时在电路单元内部可以使用该输出信号。

小提示：

OUT 与 BUFFER 信号的区别就在于信号是否往内部有反馈，将输出端口定义为 BUFFER 型，可以省去一个用于中间运算的一个临时信号，但是并不推荐这么做。

4. 端口数据类型（TYPE）定义端口信号的数据类型，在VHDL中，常用的端口信号数据类型如下：

（1） 位（BIT）型：表示一位信号的值，可以取值‘0’和‘1’，放在单引号里面表示，如X < = ‘1’，Y <= ‘0’。

（2） 位向量（BIT_VECTOR）型：表示一组位型信号值，在使用时必须标明位向量的宽度（个数）和位向量的排列顺序，例如：Q : OUT BIT_VECTOR(3 downto 0)，表示Q3，Q2，Q1，Q0四个位型信号。位向量的信号值放在双引号里面表示，例如Q <= “0000”;

（3） 标准逻辑位（STD_LOGIC）型：IEEE标准的逻辑类型，它是BIT型数据类型的扩展，可以取值‘U’，‘X’，‘0’，‘1’，‘Z’，‘W’，‘L’，‘H’，‘-’等。

（4） 标准逻辑位向量（STD_LOGIC_VECTOR）型：IEEE标准的逻辑向量，表示一组标准逻辑位型信号值。

VHDL是与类型高度相关的语言，不允许将一种数据类型的信号赋予另一种数据类型的信号。除了上述介绍的数据类型外，还有其他多种数据类型用于定义内部信号和变量，请参见1-2节。

小提示：
相同类型（模型相同，数据类型相同）的端口可以写在同一行，如：

```
ENTITY cntm16 IS
PORT(
        EN, Rd,CLK : IN STD_LOGIC;
        Co : OUT STD_LOGIC;
        Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0)
      );
END cntm16;
```

此外要注意，最后一个端口结尾没有分号！

1.1.4 VHDL语言的结构体（ARCHITECTURE）

结构体是VHDL程序设计中的最主要组成部分，是描述设计单元的具体结构和功能，在程序中，结构体放在实体的后面。每一个结构体都有名称，结构体的名称是由设计者任取的，结构体是以标识符ARCHITECTURE开头，以END结尾。结构体可以有三种描述方式，分别是行为（BEAVHER）描述方式、数据流（DATAFLOW）描述方式和结构(STRUCTURE)描述方式，其中数据流（DATAFLOW）描述方式又称为寄存器（RTL）描述方式。不同的结构体采用不同的描述语句。

结构体的一般格式为：

```
ARCHITECTURE 结构体名 OF 实体名称 IS
说明语句
BEGIN
电路描述语句
END 结构体名;
```

结构体说明语句是对结构体中用到的数据对象的数据类型、元件和子程序等加以说明。电路描述语句用并行语句来描述电路的各种功能，这些并行语句包括并行信号赋值语句、条件赋值(WHEN-ELSE)语句、进程（PROCESS）语句、元件例化(COMPONET MAP)语句和子程序调用语句等。

小提示：
结构体中定义的参数（信号，变量等）名称不能与其所属实体的端口名重名。
结构体的结束语句也可以写成 END ARCHITECTURE 结构体名,或者简写为 END。

例1-1-2设计程序的结构体部分如下：

```
ARCHITECTURE counstr OF cntm16 IS
BEGIN
  Co <= ‘1’WHEN (Q =”1111” AND EN =‘1’) ELSE ‘0’; --条件赋值语句
    PROCESS (CLK, Rd) --PROCESS语句
    BEGIN
    IF (Rd=‘0’) THEN --IF语句
    Q<= ”0000”;
    ELSIF (CLK’ EVENTAND CLK=‘1’) THEN --CLK上升沿计数
      IF(EN=‘1’) then
      Q <= Q+1;
      ENDIF;
    END IF;
  END PROCESS;
END counstr;
```

结构体的名称是counstr，该结构体属于行为描述方式，采用多种描述语句，如进程（PROCRESS）语句，条件赋值语句（WHEN-ELSE），顺序语句（IF-ELSE ）等，这些语句的具体用法参见1-3节相关内容。

> 小提示：
> 一个实体可以有多个结构体（反之不成立），多个结构体代表实体实现的多种方式，同一个实体的各结构体之间地位等同，可以采用配置语句将特定的某个结构体关联到实体，这样使同一个实体可以设计为多种实现功能，但是笔者不推荐使用多个结构体来实现实体功能，因为在综合时，配置语句是不可综合的，所以尽量每个实体仅一个结构体表述完整，这样比较清晰，整体化。所以就不介绍配置语句了，有兴趣的读者请查阅相关教材。

1.1.5 程序包（PACKAGE）、库（LIBRARY）和USE语句

程序包定义了一组标准的数据类型说明、常量说明、元件说明、子程序说明和函数说明等，它是一个用VHDL语言描写的一段程序，可以供其他设计单元调用。它如同C语言中的*.H文件一样，定义了一些数据类型说明和函数说明。在一个设计单元中，在实体部分所定义的数据类型、常数和子程序在相应的结构体中是可以被使用的（可见的），但是在一个实体的说明部分和结构体部分中定义的数据类型、常量及子程序却不能被其它设计单元的实体和结构体使用（不可见）。程序包就是为了使一组类型说明、常量说明和子程序说明对多个设计单元都可以使用而提供的一种结构。程序包分为两大类，即VHDL预定义标准程序包和用户定义的程序包。VHDL设计中常用的标准程序包的名称和内容如见表1-1-3所列。用户定义的程序包是设计者把预先设计好的电路单元设计定义在一个程序包中，放在指定的库中，以供其它设计单元调用，如果在设计中要使用某个程序包中的内容时，可以用USE语句打开该程序包。有关程序包的设计方法参见1-4-5节的内容。

库（LIBRARY）是专门用于存放预先编译好的程序包的地方，它实际上对应一个文件目录，程序包的文件就存放在此目录中。库名与目录名的对应关系可以在编译程序中指定，库的说明总是放在设计单元的最前面。例如，对IEEE标准库的调用格式为：

LIBRARY IEEE;

表 1-1-3 IEEE 两个标准库 STD 和 IEEE 中的程序包

库名	程序包名	定义的内容
STD	STANDARD TEXTIO	定义VHDL的数据类型，如BIT，BIT_VECTOR等 TEXT读写控制数据类型和子程序等
IEEE	STD_LOGIC_1164	定义STD_LOG，STD_LOGIC_VECTOR等
	STD_LOGIC_ARITH	定义有符号与无符号数据类型，基于这些数据类型的算术运算符，如“+”, “-”, “*”, “/”SHL, SHR等
	STD_LOGIC_SIGNED	定义基于STD_LOGIC与STD_LOGIC_VECTOR数据类型上的有符号的算术运算
	STD_LOGIC_UNSIGNED	定义基于STD_LOGIC与STD_LOGIC_VECTOR类型上的无符号的算术运算

1. 常用的库和包的种类

VHDL程序中常用的库有STD库、IEEE库和WORK等。其中STD和IEEE库中的标准程序包是由提供EDA工具的厂商提供的，用户在设计程序时可以用相应的语句调用。

（1）STD库

STD库是VHDL语言标准库，库中定义了STANDARD和TEXTIO两个标准程序包。STANDARD程序包中定义了VHDL的基本的数据类型，如字符（CHARACTER）、整数（INTEGER）、实数（REAL）、位型（BIT）和布尔量（BOOLEAN）等。用户在程序中可以随时调用STANDARD包中的内容，不需要任何说明。TEXTIO程序包中定义了对文本文件的读和写控制的数据类型和子程序。用户在程序中调用TEXTIO包中的内容，需要USE语句加以说明。

（2）IEEE库

IEEE标准库是存放用VHDL语言编写的多个标准程序包的目录，IEEE库中的程序包有STD_LOGIC_1164，STD_LOGIC_ARITH，STD_LOGIC_UNSIGNED和STD_LOGIC_SIGNED等程序包。其中STD_LOGIC_1164是IEEE标准的程序包，定义了STD_LOGIC和STD_LOGIC_VECTOR等多种数据类型，以及多种逻辑运算符子程序和数据类型转换子程序等。STD_LOGIC_ARITH和 STD_LOGIC_UNSINGED等程序包是SYNOPSYS公司提供的，包中定义了SIGNED和UNSIGNED数据类型以及基于这些数据类型的运算符子程序。用户使用包中的内容，需要用USE语句加以说明。

（3）WORK库

WORK库是用户进行VHDL设计的当前目录，用于存放用户设计好的设计单元和程序包。在使用该库中的内容时不需要进行任何说明。

2．库、包和USE语句的格式

用户在用到标准程序包中内容时，除了STANDARD程序包以外，都要在设计程序中加以说明，首先用LIBRARY语句说明程序包所在的库名，再用USE语句说明具体使用哪一个

程序包和具体的子程序名。各种标准程序包中的内容太多，初学者一时之间难以全面了解，可以用下面的格式，以免出现不必要的错误。

库和包的调用格式：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

小提示：

以下四个语句最好写任何模块的时候先加上，以免出现库没包含全的问题。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

1.2 VHDL的数据类型和数据对象

VHDL 语言和其它高级语言一样，除了具有一定的语法结构外，还定义了常数、变量和信号等三种数据对象，每个数据对象要求指定数据类型，每一种数据类型具有特定的物理意义。由于VHDL语言是强类型语言，不同的语句类型的数据之间不能进行运算和赋值，我们有必要详细了解VHDL语言的数据类型和数据对象。

1.2.1 VHDL的标记

一个完整的VHDL语句可以有下列几个部分组成：标识符、保留字（Reserved Words）、界符、常数、赋值符号和注释（Comments），所有这些统称为标记。

1. 标识符

标识符是程序员为了书写程序所规定的一些词，用来表示常数、变量、信号、子程序、结构体和实体等名称。VHDL基本的标识符组成的规则如下：

（1） 标识符由26个英文字母、数字0，1，2，…，9及下划线“_”组成；

（2） 标识符必须是以英文字母开头；

（3） 标识符中不能有两个连续的下划线“_”，标识符的最后一个字符不能是下划线；

（4） 标识符中的英文字母不区分大小写；

（5） 标识符字符最长可以是32个字符。

例如：

CLK，QO，DAT1，SX_1，NOT_Q是合法的标识符。

3DA，_QD，NA__C, DB-A，DB_等是非法的标识符。

小提示：

93标准定义了扩展表示符，可以以数字打头，使用VHDL保留字等，不过习惯上依然使用87标准。

2. 保留字

VHDL中的保留字是具有特殊含义的标识符号，只能作为固定的用途，用户不能用保留字作为标识符。比如ENTITY，ARCHITECTURE，PROCESS，BLOCK，BEGIN和END等。

VHDL保留字如表1-2-1所列。

表1-2-1 VHDL保留字

abs	access	after	alias	all
and	architecture	array	assert	attribute
begin	block	body	buffer	bus
case	component	configuration	constant	disconnect
downto	else	elsif	end	entity
exit	file	for	function	generate
generic	group	guarded	if	impure
in	inertial	inout	is	label
library	linkage	literal	loop	map
mod	nand	new	next	nor
not	null	of	on	open
or	others	out	package	port
postponed	procedure	process	pure	range
record	register	reject	rem	report
return	rol	ror	select	severity
signal	shared	sla	sll	sra
srl	subtype	then	to	transport
type	unaffected	units	until	use
variable	wait	when	while	with
xnor	xor			

3. VHDL中的界符

界符是作为VHDL语言中两个部分的分隔符用的。如每个完整的语句均以“;”结尾，用双减号“-”开头的部分是注释内容，不参加程序编译。信号赋值符号是“<=”，变量赋值符号是“:=”等。

在VHDL中，常用的界符如表1-2-2所列

表1-2-2 VHDL中的界符

.	:	;	>	<	=
+	–	*	/	&	<>
--	_	( )	<=	:=	

4. 注释符

在VHDL中，为了便于理解和阅读程序，常常加上注释，**注释符用双减号“–”表示**。注释语句以注释符打头，到行尾结束。注释可以加在语句结束符“;”之后，也可以加在空行处。

1.2.2 VHDL的数据类型

在VHDL中，定义了三种数据对象，即信号、变量和常数，每一个数据对象都必须具有确定的数据类型，只有相同的数据类型的两个数据对象才能进行运算和赋值，为此VHDL

定义了多种标准的数据类型，而且每一种数据类型都具有特定的物理意义。例如，BIT型、STD_LOGIC型、INTEGER型和REAL型等数据类型。

VHDL的数据类型较多，根据数据用途分类可分为标量型、复合型、存取型和文件型。标量型包括整数类型、实数类型、枚举类型和时间类型，其中位（BIT）型和标准逻辑位（STD_LOGIC）型属于枚举类型。复合型主要包括数组（ARRAY）型和记录（RECORD）型，存取类型和文件类型提供数据和文件的存取方式。这些数据类型又可以分为两大类：即在VHDL程序包中预定义的数据类型和用户自定义的数据类型。预定义的数据类型是最基本的数据类型，这些数据类型都定义在标准程序包STANARD、STD_LOGIC_1164和其它标准的程序包中，这些程序包放在EDA软件中IEEE和STD目录中，供用户随时调用。在预定义的各种数据类型的基础上，用户可以根据实际需要自己定义数据类型和子类型，如标量型和数组型。使用用户定义的数据类型和子类型可以使设计程序的语句简练易于阅读，简化设计电路硬件结构。

值得注意的是，各种EDA工具不能完全支持VHDL的所有数据类型，只支持VHDL的子集。

1. STANDARD程序包中预定义的数据类型

(1) 整数（INTEGER）数据类型

整数数据类型与数学中整数的定义是相同的，整数类型的数据代表正整数、负整数和零。VHDL整数类型定义格式为：

```
TYPE INTEGER IS RANGE -2147483648 TO 2147483647 ;
```

实际上一个整数是由32位二进制码表示的带符号数的范围。

正整数（POSITIVE）和自然数（NATURAL）是整数的子类型，定义格式为：

```
SUBTYPE POSITIVE IS INTEGER RANGE 0 TO INTEGER'HIGH ;
SUBTYPE NATURE IS INTEGER RANGE 1 TO INTEGER'HIGH ;
```

其中INTEGER'HIGH是数值类属性，代表整数上限的数值，也即2147483647。所以正整数表示的数值范围是0～2147483647，自然数表示的数值范围是1～2147483647。实际使用过程中为了节省硬件组件，常用RANGER…TO…限制整数的范围。例如：

```
SIGNALA :INTEGER; --信号A是整数数据类型
VARIABLE B :INTEGER RANGE 0 TO 15;
--变量B是整数数据类型，变化范围是0到15。
SIGNAL C :INTEGER RANGE 1 TO 7;
--信号C是整数数据类型，变化范围是1到7。
```

(2) 实数（REAL）数据类型

VHDL实数数据类型与数学上的实数相似，VHDL的实数就是带小数点的数，分为正数和小数。实数有两种书写形式即小数形式和科学计数形式，不能写成整数形式。例如

1.0，1.0E4，-5.2等实数是合法的。实数数据类型的定义格式为：

```
TYPE REAL is range -1.7e38 to 1.7e38;
```

例如：SIGNAL A, B, C :REAL ;

```
A<= 5.0;
B <= 3.5E5;
C <= -4.5;
```

小提示：
整数与实数均可以由下划线分割，便于阅读，如：45_133_134； 124_452_112.113_429；
此外不同进制的数可以由如下格式表达：**基数#数字文字#E指数**
如：2#1111_1110# = 254； 16#E#E1 = $14*16^1$ = 224；

(3) 位（BIT）数据类型

位数据类型的位值用字符‘0’和‘1’表示，将值放在单引号中，表示二值逻辑的0和1。这里的0和1与整数型的0和1不同，可以进行算术运算和逻辑运算，而整数类型只能进行算术运算。位数据类型的定义格式为：

TYPE BIT is ('0', '1');

例如：

```
RESULT : OUT BIT;
RESULT<= ‘1’;
```

将RESULT引脚设置为高电平。

(4) 位向量（BIT_VECTOR）数据类型

位向量是基于BIT数据类型的数组。VHDL位向量的定义格式为：

TYPE BIT_VECTOR is array (NATURAL range <>) of BIT;

使用位向量必须注明位宽，即数组的个数和排列顺序，位向量的数据要用双引号括起来。例如 “1010”，X “A8”。其中1010是四位二进制数，用X表示双引号里的数是十六进制数。

例如：

```
SIGNAL A :BIT_VECTOR (3 DOWNTO 0 );
A <= “1110” ;
```

表示A是四个BIT型元素组成的一维数组，数组元素的排列顺序是A3=1，A2=1，A1=1，A0=0。

(5) 布尔（BOOLEAN）数据类型

一个布尔量具有真（TRUE）和假（FALSE）两种状态。布尔量没有数值的含义，不能用于数值运算，它的数值只能通过关系运算产生。例如，在IF语句中，A>B是关系运算，如果A=3，B=2，则A>B关系成立，结果是布尔量TRUE，否则结果为FALSE。

VHDL中，布尔数据类型的定义格式为：

TYPE BOOLEAN IS (FALSE, TRUE);

(6) 字符（CHARACTER）数据类型

在STANDARD程序包中预定义了128个ASCII码字符类型，字符类型用单引号括起来，如‘A’，‘b’，’1’等，与VHDL标识符不区分大小写不同，字符类型中的字符大小写是不同的，如‘B’和‘b’不同。

(7) 字符串（STRING）

在STANDARD程序包中，字符串的定义是：

TYPE STRING is array (POSITIVE range <>) of CHARACTER;

字符串数据类型是由字符型数据组成的数组，字符串必须用双引号括起来。
例如：
CONSTANT STR1 :STRING := “Hellow world”;
定义常数ST1是字符串，初值是“Hellow world ”。

小提示：
与C语言类似，字符类型用单引号括起来，而字符串必须用双引号括起来，别弄混了。

(8) 时间（TIME）数据类型
表示时间的数据类型，一个完整的时间类型包括整数表示的数值部分和时间单位两个部分，数值和单位之间至少留一个空格，如1 ms，20 ns等。
STANDARD程序包中定义时间格式为：

```
TYPE TIME is range -9223372036854775808 to 9223372036854775807
UNITS
fs; --  飞秒
ps = 1000 fs; --  皮秒
ns = 1000 ps; --  纳秒
us = 1000 ns; --  微秒
ms = 1000 us; --  毫秒
sec = 1000 ms; --  秒
min = 60 sec; --  分
hr = 60min; --  小时
END UNITS;
```

小提示：
实数，时间类型仅用于VHDL仿真，一般综合器不支持。

2. IEEE预定义的标准逻辑位和标准逻辑位向量
(1) 标准逻辑位（STD_LOGIC）数据类型
STD_LOGIC是位（BIT）数据类型的扩展，是STD_ULOGIC数据类型的子类型。它是一个逻辑型的数据类型，其取值取代BIT数据类型的取值0和1两种数值，扩展定义了九种值，在IEEE STD1164程序包中，STD_ULOGIC和STD_LOGIC数据类型定义格式为：

```
TYPE std_ulogic IS (
        'U', -- Uninitialized
        'X', -- Forcing Unknown
        '0', -- Forcing 0
        '1', -- Forcing 1
        'Z', --High Impedance
        'W', -- Weak Unknown
        'L', --Weak 0
        'H', --Weak 1
        '-' -- Don't care
        );
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;
```

SUBTYPE std_logic IS resolved std_ulogic;

> **小提示：**
> STD_LOGIC中的数据类型必须要大写，不能使用小写字母代替，在实际的IC集成时，一般只使用'0'，'1'，'Z'，'_'四种数据类型，其余的'W'，'L'，'H'是不可综合的。

STD_LOGIC和STD_ULOGIC数据类型的区别在于STD_LOGIC数据类型是经过重新定义的，可以用来描述多路驱动的三态总线，而STD_ULOGIC数据类型只能用于描述单路驱动的三态总线。

(2) 标准逻辑位向量（STD_LOGIC_VECTOR）数据类型

STD_LOGIC_VECTOR是基于STD_LOGIC数据类型的标准逻辑一维数组，和BIT_VECTOR数组一样，使用标准逻辑位向量必须注明位宽和排列顺序，数据要用双引号括起来。

例如：

```
SIGNAL SA1 :STD_LOGIC_VECTOR (3 DOWNTO 0 );
SA1 <= “0110” ;
```

在IEEE_STD_1164程序包中，STD_LOGIC_VECTOR数据类型定义格式为：

TYPE std_logic_vector IS ARRAY (NATURAL RANGE <>) OF std_logic;

3. 其它预定义的数据类型

在STD_LOGIC_ARITH程序包中定义了无符号（UNSIGNED）和带符号（SIGNED）数据类型，这两种数据类型主要用来进行算术运算。定义格式为：

TYPE UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
TYPE SIGNED is array (NATURAL range <>) of STD_LOGIC;

(1) 无符号（UNSIGNED）数据类型

无符号数据类型是由STD_LOGIC数据类型构成的一维数组，它表示一个自然数。在一个结构体中，当一个数据除了执行算术运算之外，还要执行逻辑运算，就必须定义成UNSIGNED，而不能是SIGNED或INTEGER类型。

例如：

```
SIGNAL DAT1 :UNSIGNED（3 DOWNTO 0）;
DAT1 <= “1001”;
```

定义信号DAT1是四位二进制码表示的无符号数据，数值是9。

(2) 带符号（SIGNED）数据类型

带符号（SIGNED）数据类型表示一个带符号的整数，其最高位用来表示符号位，用补码表示数值的大小。当一个数据的最高位是0时，这个数表示正整数，当一个数据的最高位是1时，这个数表示负整数。

例如：

```
VARIABLE DB1, DB2 : SIGNED（3 DOWNTO 0）;
DB1 <= “0110” ;
DB2 <= “ 1001” ;
```

定义变量DB1是6，变量DB2是-7。

4. 用户自定义的数据类型

在VHDL中，用户可以根据设计需要，自己定义数据的类型，称为用户自定义的数据类型。利用用户自己定义数据类型可以使设计程序便于阅读。用户自定义的数据类型可以通过两种途径来实现，一种方法是通过对预定义的数据类型作一些范围限定而形成的一种新的数据类型。这种定义数据类型的方法有如下几种格式：

TYPE 数据类型名称 **IS** 数据类型名**RANGE** 数据范围；

例如：

TYPE DATA IS INTEGER RANGER 0 TO 9 ;

定义DATA是INTEGER数据类型的子集，数据范围是0～9。

SUBTYPE 数据类型名称 **IS** 数据类型名 **RANGE** 数据范围；

例如：

SUBTYPE DB IS STD_LOGIC_VECTOR（7 DOWNTO 0）;

定义DB 是STD_LOGIC_VECTOR数据类型的子集，位宽8位。

另一种方法是在数据类型定义中直接列出新的数据类型的所有取值，称为枚举数据类型。定义该种数据类型的格式为：

TYPE 数据类型名称 **IS**（取值**1**，取值**2**，…）；

例如 ：

TYPE BIT IS（’0’,’1’）;

TYPE STATE_M IS (STAT0, STAT1, STAT2, STAT3);

定义BIT数据类型，取值0和1。定义STATE_M是数据类型，表示状态变量STAT0，STAT1，STAT2，STAT3。在VHDL中，为了便于阅读程序，可以用符号名来代替具体的数值，前例中STATE_M是状态变量，用符号STAT0, STAT1, STAT2, STAT3表示四种不同的状态取值是00，01，10，11。

例如定义一个 “WEEK” 的数据类型用来表示一个星期的七天，定义格式为：

TYPE WEEK IS (SUN, MON, TUE, WED, THU, FRI, SAT);

小提示：

使用枚举数据类型定义后，综合器会自动将字符类型从0开始进行二进制编码，编码的位数由枚举元素个数决定。

5．数组（ARRAY）的定义

数组是将相同类型的单个数据元素集合在一起所形成的一个新的数据类型。它可以是一维数组（一个下标）和多维数组（多个下标），下标的数据类型必须是整数。前面介绍的位向量（BIT_VECTOR）和标准逻辑位向量（STD_LOGIC_VECTOR）数据类型都属于一维数组类型。数组定义的格式为：

TYPE 数据类型名称 IS ARRAY 数组下标的范围 OF 数组元素的数据类型；

小提示：
VHDL多维数组定义，多维数组声明即将第一维的数组作为第二维数组的元素定义即可。
TYPE 1维数据类型名称 IS ARRAY 数组下标的范围 OF 数组元素的数据类型；
TYPE 2维数据类型名称 IS ARRAY 数组下标的范围 OF 上面所定义的1维数据类型；

根据数组元素下标的范围是否指定，把数组分为非限定性数组和限定性数组两种类型。非限定性数组不具体指定数组元素下标的范围，而是用NATURAL RANGER <> 表示，当用到该数组时，再定义具体的下标范围。如前面介绍的位向量（BIT_VECTOR）和标准逻辑位向量（STD_LOGIC_VECTOR）数据类型等在程序包中预定义的数组属于非限定性数组。

例如，在IEEE程序包中定义STD_LOGIC_VECTOR数据类型的语句是

```
TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;
```

没有具体指出数组元素的下标范围，在程序中用信号说明语句指定。

例如：

```
SIGNAL DAT : STD_LOGIC_VECTOR（3 DOWNTO 0）;
```

限定性数组的下标的范围用整数指定，数组元素的下标可以是由低到高，如 0 TO 3，也可以是由高到低，如7 DOWNTO 0，表示数组元素的个数和在数组中的排列方式。

例如：

```
TYPE D IS ARRAY（0 TO 3）OF STD_LOGIC;
TYPE A IS ARRAY（4 DOWNTO 1）OF BIT;
```

定义数组D是一维数组，由四个STD_LOGIC型元素组成，数组元素的排列顺序是D(0)，D(1)，D(2)，D(3)。A数组是由四个元素组成的BIT数据类型，数组元素的排列顺序是A(4)，A(3)，A(2)，A(1)。

小提示：
对于数组数据类型，可以给一组数据多个值一起赋值：
如上例：

```
SIGNAL ARRAY1: D;
BEGIN
ARRAY1<=(‘1’,’0’,’0’,’1’);
```

6．数据类型的转换

在VHDL语言中，数据类型的定义是相当严格的，不同类型的数据是不能进行运算和赋值的。为了实现不同类型的数据赋值，就要进行数据类型的变换。变换函数在VHDL语言程序包中定义。在程序包STD_LOGIC_1164、STD_LOGITH_ARITH和STD_LOGIC_UNSIGNED中提供的数据类型变换函数如表1-2-3所列。

例如把INTEGER数据类型的信号转换为STD_LOGIC_VECTOR数据类型的方法是：

定义A, B为：

```
SIGNALA :INTEGER RANGER 0 TO 15;
SIGNALB : STD_LOGIC_VECTOR(3 DOWNTO 0);
```

需要调用STD_LOGIC_ARITH程序包中的函数CONV_STD_LOGIC_VECTOR

调用的格式是：`B <= CONV_STD_LOGIC_VECTOR(A);`

表1-2-3 数据类型变换函数

程序包名称	函数名称	功能
STD_LOGIC_1164	TO_BIT TO_BITVECTOR TO_STDULOGIC TO_STDULOGICVECTER	由STD_LOGIC转换为BIT 由STD_LOGIC_VECTOR转换为BIT_VECTOR 由BIT转换为STD_LOGIC 由BIT_VECTOR转换为STD_LOGIC_VECTOR
STD_LOGIC_ARITH	CONV_INTEGER CONV_UNSIGNED CONV_STD_LOGIC_VECTOR	由UNSIGNED, SIGNED 转换为INTEGER 由SIGNED, INTEGER转换为UNSIGNED 由INTEGER, UNSDGNED, SIGNED转换为STD_LOGIC_VECTOR
STD_LOGIC_UNSIGNED	CONV_INTEGER	由STD_LOGIC_VECTOT转换为INTEGER

1.2.3 VHDL的运算符

与高级语言一样，VHDL 语言的表达式也是由运算符和操作数组成的。VHDL 标准预定义了四种运算符，即逻辑运算符、算术运算符、关系运算符、移位运算符和连接运算符，并且定义了与运算符相应的操作数的数据类型。各种运算符之间是有优先级的，例如在所有运算符中，逻辑运算符 NOT 的优先级别最高。表 1-2-4 列出了所有运算符的优先级顺序。

表 1-2-4 VHDL 运算符列表

运算符类型	运算符	功能	优先级
逻辑运算符	AND	逻辑与	最低
	OR	逻辑或	
	NAND	逻辑与非	
	NOR	逻辑或非	
	XOR	逻辑异或	
	NXOR	逻辑异或非	
关系运算符	=	等于	
	/=	不等于	
	<	小于	
	>	大于	
	<=	小于等于	
	>=	大于等于	
移位运算符	SLL	逻辑左移	
	SLA	算术左移	
	SRL	逻辑右移	
	SRA	算术右移	
	ROL	逻辑循环左移	
	ROR	逻辑循环右移	
符号运算符	+	正	
	-	负	
连接运算符	&	位合并	

算术运算符	+	加	
	-	减	
	*	乘	
	/	除	
	MOD	求模	
	REM	求余	
	**	乘方	
	ABS	求绝对值	
逻辑非运算符	NOT	逻辑非	最高

1．逻辑运算符

在VHDL语言中定义了七种基本的逻辑运算符，它们分别是：

AND（与）、OR（或）、NOT（非）、NAND（与非）、NOR（或非）、XOR（异或）和NXOR（异或非）等。

由逻辑运算符和操作数组成了逻辑表达式。在VHDL语言中，逻辑表达式中的操作数的数据类型可以是BIT和STD_LOGIC数据类型，也可以是一维数组类型BIT_VECTOR和STD_LOGIC_VECTOR，要求运算符两边的操作数的数据类型相同、位宽相同。逻辑运算是按位进行的，运算的结果的数据类型与操作数的数据类型相同。

例如用VHDL描述逻辑表达式是Y=AB，Z=A+B+C的程序如下：

```
ENTITY loga IS
PORT(
      A, B, C   : IN STD_LOGIC;
        Y, Z    : OUT STD_LOGIC
      );
END loga;
ARCHITECTURE stra OF loga IS
    BEGIN
    Y<= A AND B;
    Z <= A OR B OR C;
 END stra;
```

例如用VHDL描述两个位向量的逻辑运算的程序如下：

```
ENTITY logb IS
PORT(
          A, B : IN BIT_VECTOR（0 TO 3）;
          Y : OUT BIT_VECTOR（0 TO 3）
          ）;
END logb;
ARCHITECTURE strb OF logb IS
     BEGIN
Y<=A AND B;
END strb;
```

如果A=1011，B=1101，则程序仿真的结果是Y=1001。

在一个逻辑表达式中有两个以上的运算符时，需要用括号对这些运算进行分组。

例如语句

```
X1 <=(A AND B ) OR (C AND B);
```

```
X2 <=( A OR B) AND C;
```

是正确的。

如果一个逻辑表达式中只有AND、OR和XOR三种运算符中的一种运算符，那么改变运算顺序不会影响电路的逻辑关系，表达式中的括号是可以省略的。例如下列语句是正确的。

```
Y1 <=A AND B AND C;
```

```
Y2 <= A OR B OR D;
```

2．算术运算符

VHDL语言定义了五种常用的算术运算符，分别是

```
+ 加或正， A+B，+A
- 减或负， A-B， -B
* 乘， A*B
/ 除， A/B
** 指数， N**2
```

以及MOD（求模）、REM（取余）、ABS（求绝对值）等算术运算符。

在算术运算表达式中，两个操作数必须具有相同的数据类型，加法和减法的操作数的数据类型可以是整数、实数或物理量，乘除法的操作数可以是整数或实数。为了节约硬件资源，除法和乘法的操作数应该选用INTEGER、STD_LOGIC_VECTOR或BIT_VECTOR等数据类型。

例如

```
X<=A+B;
Y<=C*D;
Z<=A-C**2
```

小提示：

对于 / , MOD, REM运算，要求操作符的右操作数必须为2的正整数次幂，可以用实际电路移位实现，才可以综合。

3．关系运算符

关系运算符是将两个相同类型的操作数进行数值比较或关系比较，关系运算的结果的数据类型是TRUE或FALSE，即BOOLEAN类型。VHDL语言中定义了六种关系运算符，分别是：

```
= 等于
/= 不等于
> 大于
< 小于
>= 大于或等于
<= 小于或等于
```

在VHDL中，关系运算符的数据类型根据不同的运算符有不同的要求。其中“=”（等于）和“/=”（不等于）操作数的数据类型可以是所有类型的数据，其他关系运算符可以使用整数类型、实数类型、枚举类型和数组。整数和实数的大小排序方法与数学中的比较大小方法相同。枚举型数据的大小排序方法与它们的定义顺序一致，例如BIT型数据1>0，BOOLEAN型数据TRUE>FALSE。

在利用关系运算符对位向量数据进行比较时，比较过程是从左到右的顺序按位进行比较的，操作数的位宽可以不同，但有时会产生错误的结果。

如果A、B是STD_LOGIC_VECTOR数据类型， A = “1110”，B= “10110”，关系表达式A>B的比较结果是TRUE，也就是说A>B。对于以上出现的错误可以利用STD_LOGIC_ARITH程序包中定义的数据类型UNSIGNED来解决，把要比较的操作数定义成UNSIGNED数据类型。

4．移位运算符

VHDL93标准中增加了六个移位运算符，分别是SLL逻辑左移，SRL逻辑右移，SLA算术左移，SRA算术右移，ROL逻辑循环左移，ROR逻辑循环右移。移位运算符的格式是：

操作数名称移位运算符移位位数；

操作数的数据类型可以是BIT_VECTOR、STD_LOGIC_VECTOR等一维数组，也可以是INTEGER型，移位位数必须是INTEGER型常数。

六条移位运算符所执行的操作如图1-2-1所示。

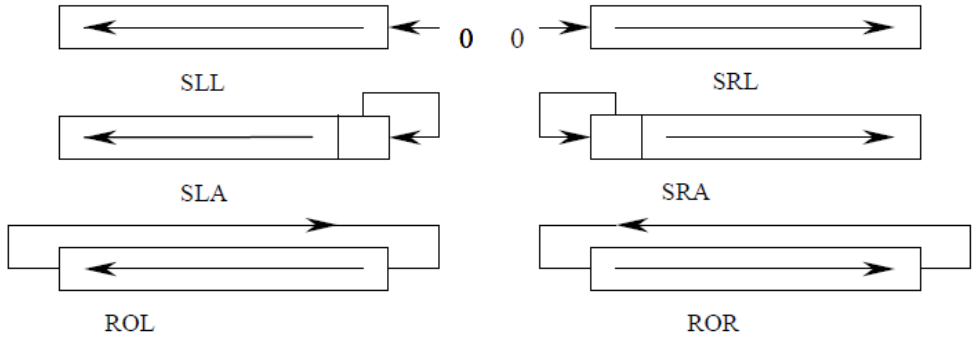


图1-2-1 移位运算符操作示意图

其中SLL是将位向量左移，右边移空位补零。SLA是将位向量左移，右边第一位的数值保持原值不变。 SRL是将位向量右移，左边移空位补零。SRA是将位向量右移，左边第一位的数值保持原值不变。ROR和ROL是自循环移位方式。

例如

```
A<= “0101”;
B <=A SLL 1;
```

仿真的结果是B = 1010。

5．连接运算符（&）

用连接运算符可以将多个数据对象合并成一个新的一维数组，也可以将两个一维数组中的元素分解合并成新的一维数组。连接两个操作符产生新的一维数组的位宽等于两个操作数的位宽之和，新的数组元素的顺序是由操作数的位置决定的，连接符“&”左边的操作数的元素在左，连接符 “&”右边的操作数的元素在右。操作数可以是BIT或STD_LOGIC数据类型。

如果

```
1011,0010 A :BIT_VECTOR（0 TO 7）
0001,0110 B :BIT_VECTOR（0 TO 7）
C <=B(0 TO 3)&A( 5 TO 7)&'1';
```

则C=1011,1101

1.2.4 VHDL的数据对象

在算法语言中，定义了多种数据对象，如常数、变量和数组等，用来存放不同类型的数据，如整数、实数、复数、逻辑常数和逻辑变量等。在VHDL程序中，常用的数据对象分为三种类型，即常数（CONSTANT）、变量（VARIABLE）和信号（SIGNAL），在使用过程中，这三种数据对象除了具有一定的数据功能外，还赋予了不同的物理意义，在应用时要特别注意。

1. 常数（**CONSTANT**）

常数被赋值后就保持某一固定的值不变。在VHDL中，常数通常用来表示计数器的模的大小、数组数据的位宽和循环计数次数等，也可以表示电源电压值的大小。常数的使用范围与其在设计程序中的位置有关，如果常数在结构体中赋值，则这个常数可供整个设计单元使用，属于全局量，如果常数在PROCESS语句或子程序中赋值，只能供进程或子程序使用，属于局部量。程序设计中使用常数有利于提高程序的可读性和方便对程序进行修改。通常常数的赋值在程序开始前进行，其数据类型在常数说明语句中指明，赋值符号为“:=”。

常数定义语句的格式为：

CONSTANT 常数名称：数据类型 := 表达式

例如：

```
CONSTANT Vcc :REAL := 5.0;
CONSTANT DALY : TIME := 20ns;
CONSTANT KN : INTEGER := 60;
```

在上面的例子中，Vcc的数据类型是实数，被赋值为5.0，DALY被赋值为时间常数20ns，KN被赋值为60的整数。

注意：常数所赋的值的数据类型必须与定义的数据类型一致，在程序中常数被赋值后不能再改变。

2．变量（VARIABLE）

在VHDL程序中，变量只能在进程和子程序中定义和使用，不能在进程外部定义使用，变量属于局部量，在进程内部主要用来暂存数据。对变量操作有变量定义语句和变量赋值语句，变量在赋值前必须通过定义，可以在变量定义语句中赋初值，变量初值不是必需的，变量初值的赋值的符号是“ :=”。

> **小提示：**
> 变量与信号赋初值语句仅可用于仿真，在综合时被忽略，不起作用。

变量定义语句的格式为：

VARABLE 变量名称：数据类型 := 初值；

例如：

```
VARABLE S1 :INTEGER := 0 ;
VARABLE S2, S3 :INTEGER ;
```

```
VARABLE CON1 :INTEGER RANGER 0 TO 20 ;
VARABLE D1, D2 :STD_LOGIC ;
```

S1是整数型变量、初值是0；CON1是整数型变量，其变化范围是0到20；D1，D2是一位标准逻辑位型变量。

变量赋值语句的格式为：

变量名称 := 表达式 ；

在对变量进行赋值时，要求表达式的数据类型必须与变量定义语句中的数据类型一致，表达式的数据对象可以是常数、变量和信号。**变量赋值是立即发生的，没有任何时间延迟的，所以变量只有当前值，并且对同一个变量可以多次赋予新值。多个变量的赋值是根据赋值语句在程序中的书写位置，按照自上而下顺序进行的，所以变量赋值语句属于顺序执行语句。变量不能放在进程的敏感信号表中。**

例如：

```
PROCESS（D, E）
VARIABLE AV, BV, CV :INTEGER := 0 ;
BEGIN
    AV := 1 ;
    BV := AV + D ;
    AV := E + 2 ;
    CV := AV * 2 ;
    A <= AV ;
    B <= BV ;
    C <= CV ;
END PROCESS ;
```

这是一个进程语句，定义AV，BV，CV是整数型变量，当敏感信号D，E只要有一个发生变化，放在进程中的语句就要全部执行一次，如D = 1，E 变化为 2，则这段程序的执行结果是：A = 4，B =2，C = 8。

3. 信号（**SIGNAL**）

在VHDL中，信号分为外部端口信号和内部信号，外部端口信号是设计单元电路的引脚，在程序实体中定义，外部信号对应四种I/O状态是IN，OUT，INOUT，BUFFER等，其作用是在设计单元电路之间起互连作用，外部信号可以供整个设计单元使用属于全局量。例如在结构体中，外部信号可以直接使用，不需要加以说明，可以通过信号赋值语句给外部输出信号赋值。

小提示：

内部信号是用来描述设计单元内部的传输信号，它除了没有外部信号的流动方向之外，其它性质与外部信号一致。

内部信号的使用范围（可见性）与其在设计程序中的位置有关，内部信号可以在包体、结构体和块语句中定义，如果信号在结构体中定义，则可以在供整个结构体中使用，如果在块语句中定义的信号，只能供块内使用，不能在进程和子程序中定义内部信号。信号在状态机中表示状态变量。

对内部信号操作有信号定义语句和信号赋值语句，内部信号在赋值前必须通过定义，可以在信号定义语句中赋初值，内部信号初值不是必需的，内部信号的定义格式与变量的

定义格式基本相同，只要将变量定义中的保留字VARIABLE换成SIGNAL即可。

内部信号定义语句的格式为：

SIGNAL 信号名称：数据类型 := 初值；

例如：

```
SIGNAL S1 :STD_LOGIC := '0' ;
SIGNAL D1 :STD_LOGIC _VECTOR（3 DOWNTO 0）:= "1001" ;
```

定义信号 S1 是标准逻辑位型，初值是逻辑 0 ；信号 D1 是标准逻辑位向量，初值是逻辑向量 1001。

信号赋值符号与变量赋值符号不同，信号赋值符号为 " <="。

小提示：

对于赋值语句还可以采用如下格式：

```
SIGNAL VEC1,VEC2: BIT_VECTOR( 3 DOWNTO 0);
VEC1<=( 0=>'1' OTHERS=> '0');-- 表示判别性的整集内涵赋值，VEC的第三位为1，其余为0
```

信号赋值语句的格式为：

信号名称 <= 表达式；

在对信号进行赋值时，表达式的数据对象可以是常数、变量和信号，但是要求表达式的数据类型必须与信号定义语句中的数据类型一致。在结构体中信号的赋值可以在进程中也可以在进程外，但两者的赋值方式是不同的。在进程外，信号的赋值是并行执行的，所以被称之为并行信号赋值语句。在进程内，信号的赋值方式具有特殊性。

信号不能在进程中定义，但可以在进程中赋值。在进程中，变量赋值是立即起作用的，信号只有在进程被激活（敏感信号发生变化）后，在进程结束时才能赋予新的值。信号具有时间特性，信号赋值不是立即发生的，需要经过固有的时间延迟，所以信号具有过去值和当前发生值，这与实际电路的特性是一致的。信号的赋值过程分为顺序处理和并行赋值两个阶段。顺序处理是按照自上而下的顺序，用信号原来的值对所有的表达式进行运算，运算结果不影响下一个表达式的运算，直到处理好进程中的最后一个表达式。并行赋值是把表达式的值并行同时赋给信号。整个过程是一个无限循环的过程，循环停止的条件是敏感信号保持不变，所以在进程中的信号赋值语句属于顺序执行语句。在进程之外的信号赋值语句属于并行同时语句。

小提示：

信号与变量赋值的区别

```
SIGNAL D :INTEGER ;                --信号要定义在进程外的结构体中
PROCESS（A,B,C）
BEGIN
        D <= A ;
        X <= B + D ;
        D <= C ;
        Y <= B + D ;
END PROCESS ;
执行的结果是:    D <= C ;
                 X <= B + C ;
                 Y <= B + C ;      --进程中同一信号多次赋值 只有最后一次生效

PROCESS（A,B,C）
VARIABLE D :INTEGER ;              --变量要定义在进程内部
BEGIN
        D := A ;
        X <= B + D ;
        D := C ;
        Y <= B+ D ;
END PROCESS ;
执行的结果是:    X <= B + A ;
                 Y <= B + C ;     --进程中变量多次赋值则立即生效
```

例1-2-3 通过一位BCD码的加法器的程序，比较信号、常量、变量的赋值及使用方法。

```
ENTITY bcdadd IS
PORT（
        op1, op2 :IN INTEGER RANGE 0 TO 9 ;
        result :OUT INTEGER RANGE 0 TO 31
      ）;
END bcdadder;
ARCHITECTURE a OF bcdadder IS
CONSTANT adj :INTEGER := 6 ; --定义常数adj=6
SIGNAL binadd :INTEGER RANGE 0 TO 18 ;
        --定义信号binadd的取值范围是0～18
BEGIN
binadd <=op1+op2; --求op1+op2和运算
PROCESS (binadd)
    VARIABLE tmp : INTEGER:=0; --定义变量tmp是整数型，初值是0
BEGIN
```

```
    IF binadd > 9 THEN --如果binadd大于9，结果要调整
    tmp := adj ; --方法是和加6，否则，结果加0。
    ELSE
    tmp := 0 ;
    END IF ;
    result <=binadd+tmp ; --给外部信号赋值
END PROCESS;
END a;
```

小提示：

常量，变量，信号的物理含义如下：

常量：电源，地，恒定逻辑值等常数。

变量：某些值的载体，存储单元，常用于描述算法。

信号：物理设计中的硬连接线，包括输入输出端口。

信号与常数相当于全局变量，变量相当于局部变量，变量只能存在于PROCESS，FUNCTION，PROCEDUCE中。不能带出PROCESS，FUNCTION，PROCEDUCE，传送出去，而信号可以。

1.3 VHDL设计的基本语句

VHDL常用语句可以分为两大类并行语句和顺序语句，在数字系统的设计中，这些语句用来描述系统的内部硬件结构和动作行为，以及信号之间的基本逻辑关系。顺序语句必须放在进程中，因此可以把顺序语句称作为进程中的语句。顺序语句的执行方式类似于普通计算机语言的程序执行方式，都是按照语句的前后排列的方式顺序执行的，一次执行一条语句，并且从仿真的角度来看是顺序执行的。结构体中的并行语句总是处于进程的外部，所有并行语句都是一次同时执行的，与他们在程序中排列的先后次序无关。

常用的并行语句有：

（1） 并行信号赋值语句，用“<=”运算符

（2） 条件赋值语句，WHEN-ELSE

（3） 选择信号赋值语句，WITH-SELECT

（4） 方块语句，BLOCK

常用的顺序语句有：

（1） 信号赋值语句和变量赋值语句

（2） IF- ELSE语句

（3） CASE-WHEN语句

（4） FOR-LOOP

1.3.1并行信号赋值语句

信号赋值语句的功能是将一个数据或一个表达式的运算结果传送给一个数据对象，这个数据对象可以是内部信号，也可以是预定义的端口信号。

例1-3-1用并行信号赋值语句描述逻辑表达式是Y=AB+C⊕D的电路。

```
ENTITY loga IS
    PORT (
```

```
        A, B, C, D : IN BIT;
          Y          : OUT BIT
        );
END loga;
--定义A，B，C，D是输入端口信号，Y是输出端口信号
ARCHITECTURE stra OF loga IS
    SIGNALE : BIT; --定义E是内部信号
    BEGIN
        Y <=(A AND B) OR E; --以下两条并行语句与顺序无关
        E <=C XOR D;
END stra;
```

小提示：
在进程中的信号赋值语句属于顺序语句，而在结构体中进程外的信号赋值语句则属于并行语句。

1.3.2 条件赋值语句，WHEN-ELSE

语法格式为：

信号Y<= 信号A WHEN 条件表达式1 ELSE
信号B WHEN 条件表达式2 ELSE
...
信号N;

在执行WHEN-ELSE语句时，先判断条件表达式1是否为TRUE，若为真，Y<=信号A，否则判断条件表达式2是否为TRUE，若为TRUE，Y<=信号B，依次类推，只有当所列的条件表达式都为假时，Y<=信号N。

例1-3-2用条件赋值语句WHEN-ELSE实现的四选一数据选择器

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY mux4 IS
PORT(
        a0, a1, a2, a3 :IN STD_LOGIC;
        s :IN STD_LOGIC_VECTOR (1 DOWNTO 0);
y :OUT STD_LOGIC
        );
END mux4;
ARCHITECTURE archmux OF mux4 IS
    BEGIN
    y <= a0 WHEN s = “00” else --当s=00时，y=a0
        a1 WHEN s = “01” else --当s=01时，y=a1
        a2 WHEN s = “10” else --当s=10时，y=a2
        a3; --当s取其它值时，y=a2
END archmux;
```

1.3.3 选择信号赋值语句，WITH-SELECT

语法格式为：

WITH 选择信号X SELECT
信号Y<= 信号A WHEN 选择信号值1,
信号B WHEN 选择信号值2,
信号C WHEN 选择信号值3,
...
信号Z WHEN OTHERS;

WITH-SELECT语句不能在进程中应用，通过选择信号X的值的变化来选择相应的操作。当选择信号X的值与选择信号值1相同时，执行Y<=信号A，当选择信号X的值与选择信号值2相同时，执行Y<=信号B，只有当选择信号X的值与所列的值都不同时，才执行Y<=信号Z。

采用选择信号赋值语句WITH-SELECT实现的四选一数据选择器结构体：

```
ARCHITECTURE archmux OF mux4 IS
BEGIN
WITH s SELECT
        y <= a0WHEN “00”,
             a1WHEN “01”,
             a2 WHEN “10”,
             a3 WHEN OTHERS;
    END archmux;
```

注意：WITH-SLECT语句必须指明所有互斥条件，即“s”的所有取值组合，因为“s”的类型为“STD_LOGIC_VECTOR”，其取值组合除了00，01，10，11外还有0x，0z，x1，…等。虽然这些取值组合在实际电路中不出现，但也应列出。为避免麻烦可以用OTHERS代替其他各种组合。

1.3.4 块（BLOCK）语句

为了实现复杂数字电路的程序设计，常常采用层次化设计和功能模块化设计方法，在VHDL语句中，实现这些功能的语句有：块语句（BLOCK），元件（COMPONENT）定义语句和元件例化（PORT MAP）语句，子程序(过程和函数)，以及包和库（LIBRARY）等。

块语句可以看作是结构体中的子模块，它把实现某一特定功能的一些并发语句组合在一起形成一个语句模块。利用多个块语句可以把一个复杂的结构体划分成多个不同功能的模块，使复杂的结构体结构分明，功能明确，提高了结构体的可读性，**块与块语句之间的关系是并行执行的**，这种结构体的划分方法仅仅只是形式上的，处于一个设计层次，块与块之间是不透明的，每个块都可以定义共块内使用的数据对象和数据类型，并且这种说明对其它块是无效的。另外，利用块语句中的保护表达式可以控制方块语句的执行。

块语句的格式为：

块标号： BLOCK
说明语句
BEGIN
并行语句区
END BLOCK 块标号;

在块语句说明部分中定义块内局部信号、数据类型、元件和子程序，在块内并行语句区可以使用VHDL中的所有并行语句。

例1-3-3设计一个电路，包含一个半加器和一个半减器，分别计算出A+B和A-B的结果。

表1-3-1半加器和半减器的真值表

输入值		半加器输出		半减法器输出	
A	B	SUM	Co	SUB	Bo
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	0	1	0
1	1	0	1	0	0

逻辑表达式：

半加器： SUM=A⊕B

Co=AB

半减法器： SUB=A⊕B

Bo=BA

把加法和减法分成两个功能模块，分别用两个BLOCK方块语句来表示，设计的程序如下所示：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY adsu is
PORT(
        a, b : IN STD_LOGIC;
        co, sum, bo, sub : OUT STD_LOGIC
        );
END adsu ;
ARCHITECTURE a OF adsu IS
BEGIN
      half_adder : BLOCK -- half_adder
      BEGIN
      sum <= A XOR B;
      co<= AAND B;
      END BLOCK half_adder;

      half_subtractor: BLOCK -- half_subtractor
      BEGIN
      sub<= a XOR b;
      bo <= NOT a AND b;
      END BLOCK half_subtractor;
END a;
```

小提示：

块语句只是起一种分隔符的作用，使程序编排更加清晰，有层次。功能上与不用块语句完全相同。

1.3.5 IF-ELSE语句

IF-ELSE语句是最常用的顺序语句，其用法和语句格式与普通的计算机高级语言类似，在VHDL语言中，它只在进程中使用，根据一个或一组条件来选择某一特定的执行通道。其常用的格式为：

格式一:

```
IF  条件表达式1 THEN
      语句方块A
ELSIF  条件表达式2 THEN
      语句方块B
ELSIF  条件表达式3 THEN
      语句方块C
      :
ELSE
     语句方块N
      END IF
```

格式二：

```
IF  条件表达式THEN
      语句方块A
END IF;
```

格式三：

```
IF  条件表达式THEN
      语句方块A
ELSE
      语句方块B
END IF
```

格式四：

```
PROCESS (CLK)
BEGIN
IF CLK'event AND CLK='1' THEN
       语句方块
      END IF;
END PROCESS;
```

语句格式一是IF语句的完整形式，格式二和格式三是IF语句的简化形式，格式四是IF语句的一种特例，它用于描述带有时钟信号CLK上升沿触发的时序逻辑电路。IF语句可以嵌套使用。IF 语句中至少应包含一个条件表达式，先判断条件表达式的结果是否为真，若为真，则执行THEN后面的语句方块的语句，执行完以后就跳转到END IF之后的语句。若条件条件表达式的结果为假，则执行ELSE之后的语句方块。

例如采用IF-ELSE实现的四选一数据选择器结构体如下：

```
ARCHITECTURE archmux OF mux4 IS
BEGIN
```

```
    PROCESS(s, a0, a1, a2, a3)
      BEGIN
      IF s= “00” THEN
          y <= a0 ;
      ELSIF s= “01” THEN
          y <= a1 ;
      ELSIF s= “10” THEN
          y <= a2;
      ELSE
          y <= a3;
      END IF;
    END PROCESS;
END archmux;
```

每一个 IF语句都必须有一个对应的 END IF语句， IF语句可以嵌套使用，即在一个IF语句中可以调用另一个 IF语句。ELSEIF允许在IF语句中出现多次。

例1-3-4用格式四描述一般的D触发器程序如下，D触发器的电路符号如图1-3-1所示。

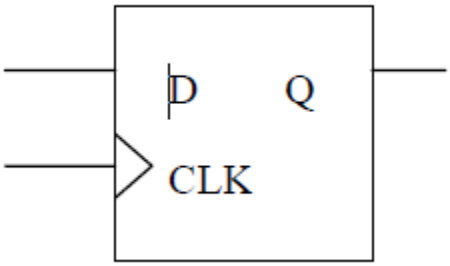


图1-3-1 D触发器

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY dff1 IS
      PORT(
      CLK, D     : IN STD_LOGIC;
      Q          : OUT STD_LOGIC
      );
      END dff1;
ARCHITECTURE a OF dff1 IS
BEGIN
      PROCESS (CLK)
      BEGIN
        IF CLK'EVENT AND CLK='1' THEN
        Q <= D;
        END IF;
      END PROCESS;
END a;
```

程序中，时钟信号（CLK）是敏感信号，用表达式CLK'EVENT AND CLK='1'判断CLK是否产生上升沿（由低电平变成高电平），若CLK产生上升沿，则执行Q <= D，否则，Q保持不变。

小提示：
判断上升沿还可以写成 IF RISING_EDGE(CLK)；如果要判断时钟信号产生下降沿，可以用表达式CLK'EVENT AND CLK='0'。或者 FALLING_EDGE(CLK)。其中CLK右上角的'表示信号的属性，其中EVENT表示信号内涵变化，是最常用的属性。对于信号的其他属性可以查阅相关书籍，由于使用并不多则不作介绍了。

1.3.6 CASE-WHEN语句

CASE-WHEN语句属于顺序语句，只能在进程中使用，常用来选择有明确描述的信号。

语法格式：

```
CASE 选择信号X IS
    WHEN 信号值1 =>
        语句方块1
    WHEN 信号值2 =>
        语句方块2
    WHEN 信号值3 =>
        :
    WHEN OTHERS =>
        语句方块 N
```

CASE-WHEN语句的功能与WITH-SELECT语句的功能相似，都是通过选择信号X的值的变化来选择相应的操作。但两者之间不同的是：

（1）CASE-WHEN语句必须放在进程中，而WITH-SELECT语句是并行语句必须放在进程外；

（2）CASE-WHEN语句根据选择信号的值，执行不同的语句方块，完成不同的功能。而WITH-SELECT语句根据选择信号的值只能执行一个操作。

(3) 使用CASE-WHEN语句时，WHEN语句中的信号值必须在选择信号的取值范围内，如果WHEN语句中列举的信号值不能覆盖选择信号S的所有取值，就用关键字OTHERS表示未能列出的其他可能的取值。

例如采用CASE-WHEN实现的四选一数据选择器结构体：

```
ARCHITECTURE archmux OF mux4 IS
  BEGIN
  PROCESS(S, A0, A1, A2, A3)
  BEGIN
  CASE S IS
          WHEN "00" => y <= A0 ;
          WHEN "01" => y<= A1 ;
          WHEN "10" => y <= A2 ;
          WHEN OTHERS => y <= A3;
END CASE;
END PROCESS;
```

```
END archmux;
```

该结构体的功能是：通过PROCESS对信号S进行感测，当S= “00”时，Y=A0，当S= “01”时，Y=A1，当S= “10”时，Y=A2，当S= “11”时，Y=A3，程序中用关键字OTHERS表示S= “11”。

> **小提示：**
> 使用CASE语句的时候要注意：
> 1. 至少要有一条分支条件
> 2. 分支条件之间不能重叠
> 3. 分支条件要覆盖完全，如果不能完全列出，则要使用OTHERS关键字。以免综合出不必要的锁存器，

1.3.7 FOR-LOOP语句

FOR-LOOP语句是一种循环执行语句，它可以使包含的一组顺序语句被循环执行，其执行的次数可由设定的循环参数决定，只要设计到重复的动作需求时，就可以考虑使用循环语句。FOR-LOOP语句分为递减方式和递增方式，两种语法格式为：

(1) 递减方式

FOR I IN 起始值 DOWNTO 结束值 LOOP
顺序语句
END LOOP；

(2) 递增方式

FOR I IN 起始值 TO 结束值 LOOP
顺序语句
END LOOP；

在循环语句中，I是循环变量决定循环次数，循环变量的变化范围由起始值和结束值的大小确定，起始值和结束值都应该取整数。当采用DOWNTO递减方式时，取起始值大于结束值，I从起始值开始，每执行一次循环后I递减1，直到结束值为止；当采用TO递增方式时，取结束值大于起始值，I也是从起始值开始执行，每次循环增加1。

例1-3-5 用FOR-LOOP语句描述奇偶校验器中的奇校验。输入四位二进制数，当检测到数据中1的位数为奇数时，输出Y=1，否则，Y=0。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY loop1 IS
PORT(
        D : IN STD_LOGIC_VECTOR(0 TO 7); --输入D是八位二进制数
         Y : OUT STD_LOGIC
         );
END loop1;

tmp := ‘0’;
BEGIN
FOR I IN 0 TO 7 LOOP
tmp := tmp XOR D(I); --变量赋值语句是立即赋值，tmp=tmp ⊕D(I)
```

```
END LOOP;
Y<= tmp ;
END PROCESS;
END a;
```

1.3.8 WHILE-LOOP语句

WHILE-LOOP循环语句，WHILE-LOOP循环是LOOP循环的另外一种形式，其语法格式为：

循环标号：while 条件表达式 loop
顺序处理语句
end loop 循环标号；

循环标号用来作为该WHILE-LOOP循环语句的标识符。WHILE后面的条件表达式是布尔表达式。WHILE-LOOP循环语句在每次执行前要先检查条件表达式的值，当条件表达式的值为TRUE时，就执行循环体中的顺序处理语句，执行完毕好返回到该循环的开始，然后再次检查条件表达式的值；若表达式值为FASLE，那么结束循环并转而执行WHILE-LOOP后的语句。

> **小提示：**
> 一般综合器不支持WHILE-LOOP语句，所以对于循环，推荐使用FOR-LOOP语句。

1.3.9 跳出循环语句

（1）NEXT跳出循环语句

在LOOP中，NEXT语句用于跳出本次循环，转入下次循环，并重新开始。其语法格式为：

NEXT 循环标号 WHEN 条件表达式；

其中循环标号用来表明结束本次循环后下一次循环的起始位置；条件表达式的值为布尔量，是跳出本次循环的条件，条件表达式的值为真时，跳出本次循环，循环标号和条件表达式是可选项。当NEXT后无循环标号和条件表达式时，要执行到该语句就立即无条件跳出本次循环，从LOOP语句的起始位置进入下一次循环。

（2）EXIT退出循环语句

在LOOP中，EXIT语句在循环体内用来描述退出循环并结束循环这一功能。其语法格式为：

EXIT 循环标号 WHEN 条件表达式；

WHEN后的条件表达式是布尔表达式，exit退出循环语句在条件表达式为真时，退出循环标号指定的循环体。

EXIT出循环语句WHEN后面的条件表达式可以缺少，成为无条件退出循环语句 **EXIT 循环标号；**

当遇到无条件退出循环语句时，立即从循环标号指明的循环体中退出，EXIT退出循环语句后的循环标号也可缺少，成为默认无条件退出循环语句时，立即从EXIT所在的循环体中退出。

> **小提示：**
> NEXT 与 EXIT 语句类似于C语言中的 CONTINUE 与 BREAK 语句。

VHDL还有WAIT语句，ASSERT语句，NULL语句，这些语句并不常用，如果要了解这些语

句的用法，请查阅相关教材。

1.4 VHDL高级语句

前面详细地介绍的 VHDL 常用的并行语句和顺序语句，在此基础上进一步介绍 VHDL 中用于结构化和模块化的设计语句，包括：进程语句（PROCESS）、元件和元件例化语句、生成语句、子程序和程序包等。

1.4.1 进程（PROCESS）语句

进程语句是在结构体中用来描述特定电路功能的程序模块。进程语句的内部主要是由一组顺序语句组成的。进程中的语句具有顺序处理和并行执行的特点。在一个结构体中可以包含多个进程语句，多个进程语句之间的是并行同时执行的，所以并行语句本身属于并行语句。进程语句即可以用来描述组合逻辑电路，也可以描述时序逻辑电路。进程语句的语法结构格式为：

<进程名称> : PROCESS <敏感信号表>
进程说明区：说明用于该进程的常数，变量和子程序。
BEGIN
变量和信号赋值语句
顺序语句
END PROCESS <进程名称>;

（1）每个进程语句结构都可以取一个进程名称，但进程语句的名称是可以选用的。进程语句从PROCESS开始至END PROCESS结束。进程中的敏感信号表（sensitivity list）只能是进程中使用的一些信号，而不能是进程中的变量。当敏感信号表中的某个信号的值发生变化时，立即启动进程语句，将进程中的顺序语句按顺序循环执行，直到敏感信号表中的信号值稳定不变为止。也可以用WAIT语句来启动进程。

（2）在进程说明部分能定义常数、变量和子程序等，但不能在进程内部定义信号，信号只能在结构体说明部分定义。

（3）在进程中的语句是顺序语句，包括信号赋值语句、变量赋值语句、IF语句、CASE语句和LOOP语句等

用PROCESS语句描述的计数器的程序如下：

```
PROCESS(CLK, Rd) --进程（敏感信号表）
BEGIN
   IF (Rd='0') THEN
   Q <= "0000";
   ELSIF (CLK' EVENT AND CLK='1') THEN
        IF(en='1') then
        Q <= Q+1;
        END IF;
   END IF;
END PROCESS;
```

在敏感信号表中，信号Rd，CLK被列为敏感信号，当此两个信号只要有一个发生变化时，此进程就被执行。注意EN并没有被列入敏感表，这是因为EN起作用必须发生在时钟的上升沿这时CLK必定发生变化，引起进程的执行。同样，若为同步清零，敏感表中可无Rd信号，此时进程如下：

```
PROCESS ( CLK ) --进程（敏感信号表）
BEGIN
```

```
    IF (CLK' EVENT AND CLK='1') THEN
        IF (Rd='0') THEN
        Q<= "0000";
        ELSIF (EN='1') then
        Q <= Q+1;
        ENDIF;
    END IF;
END PROCESS;
```

小提示：
一个结构体可以包含多个进程，他们之间是并行执行的，但是进程结构中的语句是顺序执行的。进程间通过信号来实现通信，进程信号的赋值语句是顺序执行的，但是赋值是最后并行执行的。

1.4.2 元件（COMPONET）定义语句和元件例化（PORT MAP）语句

在VHDL程序设计中，一个完整的VHDL设计程序包括实体和结构体，实体提供设计单元的端口信息，结构体描述设计单元的结构和功能，设计程序通过综合、仿真等一系列操作后，其最终的目的是得到一个具有特定功能的电路元件，因此，把这种设计好的程序定义为一个元件。这种元件可以是一个描述简单门电路的程序，也可以是一个描述一位全加器的程序，或者是其他复杂电路的描述。这些元件设计好后保存在当前工作目录中，其他设计体可以通过元件例化的方法调用这些元件。

小提示：
当前设计实体相当于一个较大（较大时相比于以前设计的实体而言）的电路系统，所声明的例化元件相当于系统板上的芯片，而当前设计实体的“端口”相当于要插入这个电路这块电路板上准备接受此芯片的一个插座。

元件（COMPONET）定义语句和元件例化（PORT MAP）语句就是用于在一个结构体中定义元件和实现元件调用的两条语句，两条语句分别放在一个结构体中的不同的位置，元件定义（COMPONET）语句放在结构体的ARCHITECTURE和BEGIN之间，指出该结构体调用哪一个具体的元件，元件调用时必须要进行数据交换，元件例化语句中的PORT MAP是端口映射的意思，表示结构体与元件端口之间交换数据的方式。其语法结构格式为：

（1）元件定义（COMPONET）语句的格式为：

COMPONENT 元件名称 IS
GENERIC 常量定义信息 （同该元件源程序实体中的GENERIC部分）
PORT 元件端口信息（同该元件源程序实体中的PORT部分）
END COMPONET;

小提示：
元件声明与实体声明基本一致，元件就是一个实体。

（2）元件例化（PORT MAP）语句的格式为：

例化名：元件名称 PORT MAP 元件端口列表

例1-4-1用元件定义（COMPONET）和元件例化语句实现四位全加器的程序设计,调用的元件是一位全加器，元件名称是fulladder，用VHDL描述的程序的文件名是fulladder.VHD，四位全加器电路图如图1-4-1所示。

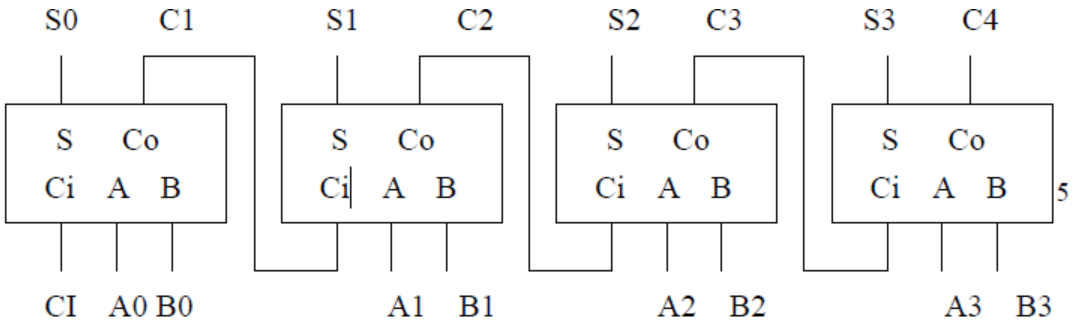


图 1-4-1 四位全加器电路图

四位全加器的程序文件名为adder4.VHD，内容如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY adder4 IS
PORT (
        A, B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        CI : IN STD_LOGIC;
        S : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        C : BUFFER STD_LOGIC_VECTOR(4 DOWNTO 1)
    );
END adder4;
ARCHITECTURE a OF adder4 IS
COMPONENT fulladder --元件定义，fulladder是元件名称
PORT( --端口名表
        A, B, Ci : IN STD_LOGIC;
        Co, S : OUT STD_LOGIC
    );
END COMPONENT;
BEGIN
U0: fulladder PORT MAP (A(0),B(0),CI,C(1),S(0)); --元件例化
U1: fulladder PORT MAP (A(1),B(1),C(1),C(2),S(1));
U2: fulladder PORT MAP (A(2),B(2),C(2),C(3),S(2));
U3: fulladder PORT MAP (A(3),B(3),C(3),C(4),S(3));
END a;
```

在上面程序的元件定义（COMPONET）语句中，COMPONET语句后面的元件名称是fulladder，其在PORT中的端口信号信息与描述一位全加器的程序名fulladder.VHD在实体中PORT部分的端口信号信息必须相同，包括端口信号名称、端口信号的输入/输出状态和端口信号的数据类型等。COMPONET语句放在结构体的ARCHITECTURE和BEGIN之间。

在PORT MAP部分，元件名称fulladder必须与COMPONT中的元件名称一致，U0、U1、U2、U3是四个元件例化名，表明在这个结构体中对fulladder单元的四次不同的调用。

PORT MAP中所列的端口信号列表表示当前设计单元与元件的端口连接方式，在VHDL设计中有两种连接方式，一种是位置关联方式，如

U0: Fulladder PORT MAP (A(0),B(0),Ci,C(1),S(0));

PORT MAP列出的端口信号名与COMPONET中的PORT端口信号名称在顺序、端口状态和数据类型上必须一致，各个端口信号的意义取决于它的位置而不是它的名称；另一种是端口信号名称关联方式，在这种关联方式下，用符号 “=>”连接元件端口和设计电路的端口，如上例U1中， PORT MAP列出的端口信号名称是A(1)，B(1)，C(0)，C(1)和S(1)，PORT MAP语句可以写成如下形式：

U1: Fulladder PORT MAP (A=>A(1), B=>B(1), Ci=>C(1), Co=>C(2), S=>S(1));

这时，各个端口的意义取决于端口的名称，与位置无关。

元件例化语句与BLOCK语句一样属于并行语句，但是元件和元件例化在设计项目中是分层次的，每个元件就是一个独立的设计实体，这样就可以把一个复杂的设计实体划分成多个简单的元件来设计。

1.4.3 生成(GENERATE)语句

生成语句是一种循环语句，具有复制电路的功能。当设计一个由多个相同单元模块组成的电路时，就可以用生成语句来描述。生成语句有FOR-GENERATE和IF-GENERATE两种形式，分别说明如下：

（1）FOR-GENERATE语句格式为：

标号：FOR 循环变量 IN 取值范围 GENERATE
并行语句
END GENERATE [标号];

FOR-GENERATE语句与FOR-LOOP语句不同，FOR-GENERATE中所列的语句是并行信号赋值语句、元件例化、块语句和子程序等并行语句。循环变量是一个局部变量，其取值范围可以选择递增和递减两种形式，如0 TO 5和3 DOWNTO 1等。生成语句所复制的单元模块是按照一定的顺序排列的，而单元模块之间的关系却是并行的。所以生成语句属于并行语句。

例1-4-2用FOR-GENERATE语句实现四位全加器的程序设计如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY adder4 IS
    PORT(
        A, B    : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        Ci      : IN STD_LOGIC;
        S       : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        C       : BUFFER STD_LOGIC_VECTOR(4 DOWNTO 0)
        );
    END adder4;
    ARCHITECTURE a OF adder4 IS
      COMPONENT fulladder
      PORT (    A, B, Ci : IN STD_LOGIC;
```

```
            Co, S : OUT STD_LOGIC);
    END COMPONENT;
    BEGIN
        C(0) <= Ci;

    gen1: FOR I IN 0 TO 3 GENERATE
    addx: fulladder PORT MAP (A(I),B(I),C(I),C(I+1),S(I)); --产生四位串行全加器
    END GENERATE ;
END a;
```

例1-4-3 用生成语句描述用四个D触发器组成一个四位移位寄存器，电路图如图1-4-2所示。D触发器用元件定义和元件例化语句调用。元件名称是dff1，用VHDL描述的D触发器的程序参见例1-3-4，程序的文件名是dff1.VHD。

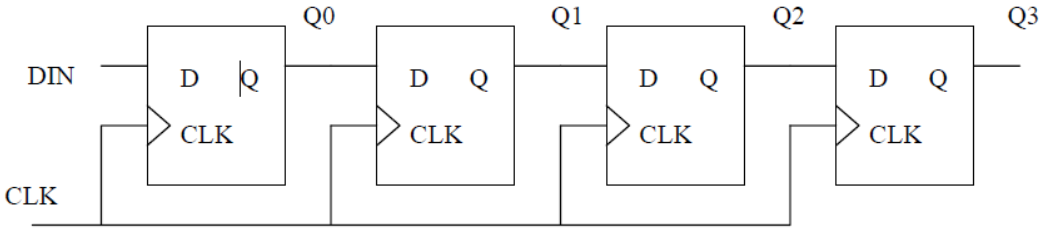


图1-4-2 四位移位寄存器

四位移位寄存器的文件名为shift.VHD，内容如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY shift IS
    PORT(
        DIN, CLK : IN STD_LOGIC;
        DOUT : OUT STD_LOGIC;
        Q : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
    END shift;
ARCHITECTURE B OF shift IS
    COMPONENT dff1
    PORT ( D, CLK    : IN STD_LOGIC;
           Q         : OUT STD_LOGIC );
    END COMPONENT;
    SIGNAL D : STD_LOGIC_VECTOR(0 TO 4);
    BEGIN
        D（0）<= DIN;
gen2: FOR I IN 0 TO 3 GENERATE
fx: dff1 PORT MAP (D(I),CLK,D(I+1));
```

```
END GENERATE ;
        Q(0) <= D(1);
        Q(1) <= D(2);
        Q(2) <= D(3);
        Q(3) <= D(4);
        DOUT<=D(4)
END B;
```

小提示：
从上例可以看出，FOR-GENERATE用来处理规则的单元模块，对于不规则的单元模块用IF-GENERATE格式。

（2）IF-GENERATE语句带有条件选择项，其格式为：

标号： IF 条件 GENERATE
并行语句
END GENERATE [标号];

例1-4-4用IF-GENERATE语句描述的四位移位寄存器，其结构体部分程序如下：

```
ARCHITECTURE C OF shift IS
COMPONENT DFF1
PORT (
        DIN, CLK : IN STD_LOGIC;
        Q : OUT STD_LOGIC
        );
END COMPONENT;
SIGNAL D : STD_LOGIC_VECTOR(0 TO 4);
BEGIN
gen3: FOR I IN 0 TO 3 GENERATE
IF I = 0 GENERATE
        fx: dff1 PORT MAP (DIN,CLK,D(I+1));
        END GENERATE ;
        IF I /= 0 GENERATE
        fx: dff1 PORT MAP (D(I),CLK, D(I+1));
        END GENERATE ;
END GENERATE ;
Q(0) <= D(1);
Q(1) <= D(2);
Q(2) <= D(3);
Q(3) <= D(4);
END B;
```

1.4.4 子程序（SUBPROGRAM）

子程序是一个VHDL程序模块，它是由一组顺序语句组成的。主程序调用子程序，子程序将处理结果返回给主程序，其含义与其它高级计算机语言中的子程序相同。子程序可以在程序包、结构体和进程中定义，子程序必须在被定义后才能被调用，主程序和子程序之间通

过端口参数列表位置关联方式进行数据传送，子程序可以被多次调用完成重复性的任务。在VHDL中的子程序有两种类型：过程（Procedure）和函数（Function），他们在被调用后返回数据的方式不同。

小提示：
过程可以具有多个返回值，而函数只能返回一个。

1. 过程语句

过程语句的格式为：

PROCEDURE 过程名称参数列表 --过程首
PROCEDURE 过程名称参数列表 IS --过程体
说明部分
BEGIN
顺序语句
END 过程名称；
调用过程语句的格式为：
过程名称 参数列表；

在VHDL中，过程定义由两部分组成，即过程首和过程体，在进程或结构体中过程首可以省略，过程体放在结构体的说明部分。而在程序包中必须定义过程首，把过程首放在程序包的包首部分，而过程体放在包体部分。

在PROCEDURE结构中，参数列表中的参数可以是输入也可以是输出，在参数表中可以对常数、变量和信号三类数据对象作出说明，用IN、OUT和INOUT定义这些参数的端口模式，在没有特别的指定时，端口模式IN的参数默认常数，端口模式OUT和INOUT看作变量。**在子程序调用时，IN和INOUT的参数传送数据至子程序，子程序调用结束返回时，OUT和INOUT的参数返回数据。**

例1-4-5用一个过程语句来实现数据求和运算程序。

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY PADD IS
PORT(
    A, B, C          : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    CLK, SET         : IN STD_LOGIC;
          D          : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END PADD;
ARCHITECTURE a OF PADD IS

--定义过程体
PROCEDURE ADD1 (DATAA, DATAB, DATAC : IN STD_LOGIC_VECTOR;
                DATAOUT : OUT STD_LOGIC_VECTOR ) IS
    BEGIN
    DATAOUT := DATAA+DATAB+DATAC;
```

```
END ADD1;

BEGIN
  PROCESS (CLK)
    VARIABLE TMP :STD_LOGIC_VECTOR (3 DOWNTO 0);
    BEGIN
      IF (CLK'EVENT AND CLK ='1') THEN
          IF(SET= '1') THEN
          TMP:= "0000";
          ELSE
          ADD1 (A, B, C, TMPTMP);          --过程调用
          END IF;
      END IF;
      D <= TMP;
    END PROCESS;
END a;
```

2. 函数语句

函数语句分为两个部分，函数首和函数体。

（1）函数首的格式为：

FUNCTION 函数名称（参数列表）

RETURN 数据类型名；

（2）函数体的格式为：

FUNCTION 函数名称（参数列表）

RETURN 数据类型名IS

说明部分

BEGIN

顺序语句

RETURN 返回变量

END 函数名称；

在进程或结构体中函数首可以省略，而在程序包中必须定义函数首，放在程序包的包首部分，而函数体放在包体部分。

函数语句中参数列表列出的参数都是输入参数，在参数表中可以对常数、变量和信号三类数据对象作出说明，默认的端口模式是IN，在函数语句中如果参数没有定义数据类型就看作常数处理。**调用函数语句的返回数据和返回数据的数据类型分别是由RETURN后的返回变量和返回变量的数据类型决定的。**

调用函数语句的格式为：

Y<= 函数名称（参数列表）；

小提示：
子程序相当于C语言的自定义函数（函数相当于有返回值的函数，过程相当有输出参数的函数），可以在结构体的进程中进行直接调用，使语句更加简洁明了。但是也有区别：VHDL中每调用一次子程序就相当于生成了一个相应的电路模块，所以不能无限制调用子程序，而是要严格控制调用次数。

1.4.5 程序包的设计

在VHDL中，为了使已定义的数据类型、子程序和元件等被其他设计程序所利用，用户可以自己设计一个程序包，将它们收集在程序包中。程序包分为两个部分，即包首和包体两个部分，结构为：

（1）包首部分

PACKAGE 程序包名称 IS
包首说明
END 程序包名称；

（2）包体部分

PACKAGE BODY 程序包名称 IS
包体说明语句
END 程序包名称；

包首说明部分定义数据类型、元件和子程序等。包体说明语句部分具体描述元件和子程序的内容。在程序包结构中，程序包体不是必需的，因为在程序包首也可以具体定义元件和子程序的内容。

在例1-4-6中，定义一个MAX函数在程序包文件bf1.VHD中，在smax.VHD程序中用函数调用的方式直接调用包中的内容。

例1-4-6用FUNCTION语句描述在两个数中找出最大值，并用函数调用方式求出最大值。这段程序放在一个程序包（PACKAGE）中。

（1）在程序包名称为BF1的程序包中定义函数名称为MAX1的函数，程序包文件名为bf1.VHD，放在当前的WORK库中。

```
LIBRARYIEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE BF1 IS                          --定义程序包的包头，BF1是程序包名称
FUNCTION MAX1(A : STD_LOGIC_VECTOR; --定义函数首，函数名称是MAX
              B : STD_LOGIC_VECTOR)
RETURN STD_LOGIC_VECTOR;        --定义函数返回值的类型
END BF1;

PACKAGE BODY BF1 IS                 --定义程序包体
FUNCTION MAX1 (A: STD_LOGIC_VECTOR; --定义函数体
              B : STD_LOGIC_VECTOR)
RETURN STD_LOGIC_VECTOR IS
VARIABLE TMP : STD_LOGIC_VECTOR (A'RANGE);
    --属性A'RANGE表示A的数组范围，该例中A'RANGE=3 DOWNTO 0
BEGIN
```

```
        IF (A>B) THEN
            TMP:=A;
        ELSE
            TMP:=B;
        END IF;
            RETURN   TMP;        --TMP是函数返回变量
    END MAX1;
    END BF1;
```

（2）调用函数MAX1的程序，文件名是SMAX.VHD

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
LIBRARY WORK;                          --用户当前工作库，可以不列出
USE WORK.BF1.ALL;

ENTITY SMAX IS
PORT(
        D1, DA, DB : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        CLK, SET : IN STD_LOGIC;
         Do : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
END SMAX;

ARCHITECTURE a OF SMAX IS
BEGIN
    PROCESS (CLK)
    BEGIN
        IF (CLK'EVENT AND CLK ='1') THEN
            IF(SET= '1') THEN                          --SET=1，同步置数
            DO <= D1;
            ELSE
            DO <= MAX1(DA, DB);                        --调用MAX1函数
            END IF;
        END IF;
    END PROCESS;
END a;
```

小提示：

程序包是由VHDL语言编写的，所以其源程序要以XXX.VHD文件类型保存。若要使用程序包中声明的内容，在设计实体的开始，要使用USE WORK.XXX.ALL打开程序包，再调用程序包的内容；

相关信息

关于其他的相关信息，请访问以下网站

■ 心得交流与问题互助：

http://www.zr-tech.com/bbs

■ ZRtech之FPGA学友会 EDN小组欢迎您的加入

http://group.ednchina.com/2762/

■ ZRtech FPGA/CPLD普及风暴 EDN助学活动火热进行中

http://group.ednchina.com/2762/40844.aspx