第2章
VHDL 设计入门

要 求

掌握 VHDL 语言的基本知识和运用 VHDL 语言设计逻辑电路的基本方法

知识点

- 理解 VHDL 程序的基本结构
- 理解 VHDL 程序的顺序语句
- 理解 VHDL 程序的并行语句
- 理解 VHDL 程序的语言要素

重点和难点

- VHDL 程序的顺序结构
- VHDL 程序的并行结构

引 言

HDL 文本输入设计法是 Quartus Ⅱ的一个重要输入设计法，本书主要介绍 VHDL 文本输入设计法。本章讨论 VHDL 语言的基本知识和运用 VHDL 语言设计逻辑电路的基本方法，在以后各章再对 VHDL 文本输入设计法及其应用作进一步的深入讨论。

本章先介绍 VHDL 程序的基本结构、顺序语句和并行语句，最后对 VHDL 程序的语言要素做一个总结。

2.1 VHDL 程序的基本结构

VHDL 程序包含实体（ENTITY）、结构体（ARCHITECTURE）、配置（CONFIGURATION）、程序包（PACKAGE）、库（LIBRARY）5 个部分。

简单的实体是由实体和结构体两部分组成的。实体用于描述设计系统的外部接口信号，结构体用于描述系统的行为、系统数据的流程或者系统组织结构形式。设计实体是 VHDL 程序的基本单元，是电子系统的抽象。简单的实体可以是一个与门电路（AND GATE），复杂的实体可以是一个微处理器或一个数字电子系统。实体由实体说明和结构体说明两部分组成。

【例 2.1】以下是一个简单的 VHDL 源程序，它实现了一个与门。由这个程序可以归纳出 VHDL 程序的基本结构。

```
ENTITY   myand2   IS                          --实体名称为 and2
  PORT   (a,b:IN BIT;                        --a、b 是两个输入引脚
          c:OUT   BIT);                      --c 为输出引脚
  END   myand2;
ARCHITECTURE   exam1   OF   myand2   IS       --结构体 exam1 是对实体
BEGIN                                        and2 的内部描述，描述
      c<=a   AND   b;                        了 and2 器件的内部功能
                                             为实现一个 2 输入与门
END   exam1;
```

由例 2.1 可以看出，VHDL 程序由两部分组成：第 1 部分为实体说明，第 2 部分为结构体。VHDL 程序结构更抽象、更基本、更简练的表示如图 2–1 所示。

设计实体用关键字 ENTITY 来标识，结构体由 ARCHITECTURE 来标识。系统设计中的实体提供该设计系统的公共信息，结构体定义了各个模块内的操作特性。一个设计实体必须包含一个结构体或含有多个结构体。一个电子系统的设计模型如图 2–2 所示。

图 2–1　VHDL 程序基础

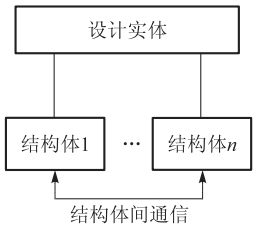


图 2–2　VHDL 程序设计系统模型

一、实体

实体由实体名、类型说明、端口说明、实体说明部分和实体语句部分组成。

1. 实体语句结构

根据 IEEE 标准，实体组织的一般格式为：

ENTITY　实体名 IS
[GENERIC(类型说明);]
[PORT(端口说明);]
实体说明部分;
[BEGIN
实体语句部分;]
END [ENTITY] [实体名];

实体名是设计者自己给设计实体的命名，其他设计实体可对该设计实体进行调用。中间在方括号内的语句描述，在特定的情况下并非是必须的。

2. 类型说明

类型说明是实体说明中的可选项，放在端口说明之前，其一般书写格式为：

GENERIC　[CONSTANT]名字表：[IN]子类型标识[:＝静态表达式],…]

举例：

GENERIC(m:TIME:＝3ns)

这个参数说明是指在 VHDL 程序中，结构体内的参数 m 的值为 3ns。

类型说明和端口说明是实体说明的组成部分，用于说明设计实体和外部通信的通道。利用外部通信通道，参数的类型说明为设计实体提供信息。参数的类型用来规定端口的大小、I/O 引脚的指派、实体中子元件的数目和实体的定时特性等信息。

3. 端口说明

端口说明是对设计实体与外部接口的描述，是设计实体和外部环境动态通信的通道，其功能对应于电路图符号的一个引脚。实体说明中的每一个 I/O 信号被称为一个端口，一个端口就是一个数据对象。端口可以被赋值，也可以当做变量用在逻辑表达式中。定义实体的一组端口称作端口说明。

端口说明的组织结构必须有一个名称、一个通信模式和一个数据类型。端口说明的一般格式为：

Port(端口名,端口名:模式　数据类型名
⋮
端口名,端口名:模式　数据类型名);

端口名是赋于每个外部引脚的名称，名称的含义要明确，如 D 开头的端口名表示数据，A 开头的端口名表示地址等。端口名通常用几个英文字母或一个英文字母加数字表示。下面是合法的端口名：CLK，RESET，A0，D3。

模式用来说明数据、信号通过该端口的传输方向。端口模式有 IN、OUT、BUFFER、INOUT。

（1）输入（IN）

输入仅允许数据流入端口。输入信号的驱动源由外部向该设计实体内进行。

输入模式主要用于时钟输入、控制输入（如 RESET、ENABLE、CLK）和单向的数据输入，如地址信号（ADDRESS）。不用的输入一般接地，以免浮动引入干扰噪声。

（2）输出（OUT）

输出仅允许数据流从实体内部输出。端口的驱动源是由被设计的实体内部进行的。输出模式不能用于被设计实体的内部反馈，因为输出端口在实体内不能看做可读的。输出模式常用于计数输出、单向数据输出、设计实体产生的控制其他实体的信号等。一般而言，不用的输出端口不能接地，避免造成输出高电平时烧毁被设计实体。

（3）缓冲（BUFFER）

缓冲模式的端口与输出模式的端口类似，只是缓冲模式允许内部引用该端口的信号。缓冲端口既能用于输出，也能用于反馈。

缓冲端口的驱动源可以是：

- 设计实体的内部信号源；
- 其他实体的缓冲端口。

缓冲不允许多重驱动，不与其他实体的双向端口和输出端口相连。

内部反馈的实现方法有：

- 建立缓冲模式端口；
- 建立设计实体的内部节点。

缓冲模式用于在实体内部建立一个可读的输出端口，例如计数器输出，计数器的现态被用来决定计数器的次态。实体既需要输出，又需要反馈，这时设计端口模式应为缓冲模式。

（4）双向模式（INOUT）

双向模式可以代替输入模式、输出模式和缓冲模式。

在设计实体的数据流中，有些数据是双向的，数据可以流入该设计实体，也有数据从设计实体流出，这时需要将端口模式设计为双向端口。

双向模式的端口允许引入内部反馈，所以双向模式端口还可以作为缓冲模式用。由上述分析可见，双向端口是一个完备的端口模式。

一般而言，输入信号把端口指派成输入模式，输出信号把端口指派成输出模式，而双向数据信号，如计算机的 PCI 总线的地址/数据复用总线、DMA 控制器数据总线，都选用端口双向模式。这一良好的设计习惯，使得从端口名称、端口模式就可一目了然地知道信号的用途、性质、来源和去向，十分方便。对一个大型设计任务，大家应协同工作，从而不至于引起歧义。

二、结构体

结构体具体指明了该设计实体的行为，定义了该设计实体的功能，规定了该

设计实体的数据流程，指派了实体中内部元件的连接关系。用 VHDL 语言描述结构体有 4 种方法：

① 行为描述法：采用进程语句，顺序描述被称设计实体的行为。

② 数据流描述法：采用进程语句，顺序描述数据流在控制流作用下被加工、处理、存储的全过程。

③ 结构描述法：采用并行处理语句描述设计实体内的结构组织和元件互连关系。

④ 采用多个进程（PROCESS）、多个模块（BLOCKS）、多个子程序（SUBPROGRAMS）的子结构方式。

结构体的一般书写格式为：

```
ARCHITECTURE 结构体名 OF 实体名 IS
定义语句，内部信号，常数，数据类型，函数定义
BEGIN
[并行处理语句];
[进程语句];
⋮
END 结构体名;
```

一个结构体的组织结构从“ARCHITECTURE 结构体名 OF 实体名 IS”开始，到“END 结构体名”结束。

结构体名称由设计者自由命名，是结构体的唯一名称。OF 后面的实体名称表明该结构体属于哪个设计实体，有些设计实体中可能含有多个结构体。这些结构体的命名可以从不同侧面反映结构体的特色，让人一目了然。例如：

```
ARCHITECTURE behacvioral OF mux IS    --用结构体行为命名
ARCHITECTURE dataflow OF mux IS       --用结构体的数据流命名
ARCHITECTURE structural OF mux IS     --用结构体的组织结构命名
ARCHITECTURE bool OF mux IS           --用结构体的数学表达方式命名
ARCHITECTURE latch OF mux IS          --用结构体的功能来定义
```

上述几个结构体都属于设计实体 mux，每个结构体有着不同的名称，使得阅读 VHDL 程序的人能直接从结构体的描述方式了解功能，定义电路行为。因为用 VHDL 写的文档不仅是 EDA 工具编译的源程序，而且最初主要是项目开发文档，供开发商、项目承包人阅读的。这就是硬件描述语言与一般软件语言不同的地方之一。

三、程序包

实体中定义的各种数据类型、子程序和元件调用说明只能局限在该实体内或结构体内调用，其他实体不能使用。出于资源共享的目的，VHDL 提供了程序包

的机制。程序包如同公用的“工具箱”，各种数据类型、子程序等一旦放入程序包，就称为共享的“工具”，各个实体都可使用程序包定义的“工具”。

程序包的语句格式如下：

```
PACKAGE   程序包名   IS                    --程序包首开始
[程序包首说明语句]
END   程序包名;                            --程序包首结束
PACKAGE   BODY   程序包名 IS               --程序包体开始
[程序包体说明语句]
END   程序包名;                            --程序包体结束
```

STD_LOGIC_1164 程序包是 IEEE 库中最常用的程序包，是 IEEE 的标准程序包。其中包括了一些数据类型、子类型和函数的定义，这些定义将 VHDL 扩展为一个能描述多值逻辑（即除具有“0”和“1”以外还有其他的逻辑量，如高阻态“Z”、不定态“X”等）的硬件描述语言。

程序包结构中，程序包体并非是必须的，程序包首可以独立定义和使用。

例如：

```
PACKAGE   PAC1   IS                                --程序包首开始
TYPE   BYTE   IS   RANGE   0   TO   255;          --定义数据类型 BYTE
SIGNAL   ADDEND : NIBBLE;                         --定义信号 ADDEND
⋮
END   PAC1;                                       --程序包首结束
```

如果要使用这个程序包中的所有定义，可用 USE 语句访问此程序包：

```
USE   WORK.PAC1.ALL;
```

四、库

在利用 VHDL 进行工程设计中，为了提高设计效率以及使设计遵循某些统一的语言标准或数据格式，有必要将一些有用的信息汇集在一个或几个库中以供调用。这些信息可以是预先定义的数据类型、子程序等设计单元的集合体（程序包），或预先设计好的各种设计实体（元件库程序包）。因此，可以把库看成是一种用来存储预先完成的程序包和数据集合体的仓库。

库的语句格式如下：

```
LIBRARY    库名;
```

这一语句相当于为其后的设计实体打开了以次库名命名的库，以便设计实体可以利用其中的程序包。如语句“LIBRARY   IEEE;”表示打开 IEEE 库。

1. 库的种类

（1）IEEE 库

IEEE 库是 VHDL 设计中最常见的库，包含有 IEEE 标准的程序包和其他一些

支持工业标准的程序包。IEEE 库中的标准程序包主要包含 STD_LOGIC_1164，STD_LOGIC_ARITH，STD_LOGIC_SIGNED，STD_LOGIC_UNSIGNED 程序包。

（2）STD 库

STD 库是 VHDL 的标准库，VHDL 在编译过程中自动使用这个库，所以使用时不需要语句说明，类似“LIBRARY　STD”这种语句是不必要的。

（3）WORK 库

WORK 库是用户的 VHDL 设计的现行工作库，用于存放用户设计和定义的一些设计单元和程序包。因此，自动满足 VHDL 的语言标准，实际调用中，也不需要语句说明。

2. 库的用法

在 VHDL 中，库的说明语句是放在实体单元的前面，而且库语言一般必须与 USE 语言同用。库语言关键词 LIBRARY，指明所使用的库名。USE 语句指明库中的程序包。一旦说明了程序包，整个设计实体都可以进入访问和调用，但其作用范围仅限于所说明的设计实体。

USE 语句的使用将使说明的程序包对本设计实体部分全部开放，USE 语句的使用有两种常用格式：

USE 库名.程序包名.项目名;

USE 库名.程序包名.ALL;

例如：

```
LIBRARY　IEEE;                          --打开 IEEE 库
USE IEEE.STD_LOGIC_1164.ALL;      --使用 IEEE 库内的 STD_LOGIC_1164
USE IEEE.STD_LOGIC_UNSIGNED.ALL; --使用 IEEE 库内的 STD_LOGIC_
                                                UNSIGNED 程序包内的所有资源
```

五、配置

配置可以把特定的结构体指定给一个确定的实体。为了满足不同设计阶段或不同场合的需要，对某一个实体，我们可以给出几种不同形式的结构体描述，这样在其他实体调用该实体时，就可以根据需要选择其中某一个结构体，选择不同的结构体以便进行性能对比试验，确认性能最佳的结构体。以上这些工作就由配置来完成。

配置语句的一般语句如下：

```
CONFIGURATION　配置名　OF　实体名　IS
[配置语句说明]
END　配置名;
```

【例 2.2】在描述一个与非门 NAND 的设计实体中会有两个不同的逻辑描述方法构成的结构体。用配置语句来为特定的结构体需求作配置指定。

```
LIBRARY  IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY  MYNAND  IS
   PORT (A:IN STD_LOGIC;
         B: IN STD_LOGIC;
         C: OUT STD_LOGIC);
END  ENTITY  MYNAND;
ARCHITECTURE  ART1  OF  MYNAND  IS
   BEGIN
   C<=NOT(A  AND  B);
END  ARCHITECYURE  ART1;
ARCHITECTURE  ART2  OF  MYNAND  IS
   BEGIN
   C<='1'  WHEN (A='0' )  AND  ( B='0')  ELSE
       '1'  WHEN (A='0' )  AND  ( B='1')  ELSE
       '1'  WHEN (A='1' )  AND  ( B='0')  ELSE
       '0'  WHEN (A='1' )  AND  ( B='1')  ELSE
       '0';
END  ARCHITECTURE;
CONFIGURATION  SECOND  OF  IS
   FOR  ART2
   END  FOR;
END  SECOND;
CONFIGURATION  FIRST  OF  IS
   FOR  ART1
   END  FOR;
END  FIRST;
```

在本例中，若指定配置名为 SECOND，则为实体 NAND 配置的结构体为 ART2；若指定配置名为 FIRST，则为实体 NAND 配置的结构体为 ART1。两种结构的描述方式不同，但是逻辑功能相同。

2.2 VHDL 程序的顺序语句

顺序语句是 VHDL 程序设计中很重要的语句系列之一，它能够从多个侧面完整地描述数字系统的硬件结构和基本逻辑功能。

顺序语句与传统软件编程语言中的语句执行方式十分相似。所谓的顺序，主

要指的是语句的执行（指仿真执行）顺序与它们书写顺序基本一致。但应注意的是，这里的顺序是从仿真软件的运行或顺应 VHDL 语法的编程思路而言的，其相应的硬件逻辑工作方式未必如此。

顺序语句只能出现在进程（PROCESS）或子程序（PROCEDURE）、函数（FUNCTION）中使用，按程序书写的顺序自上而下、一个一个语句地执行。进程本身属于并行语句，这就是说在同一设计实体中，所有的进程都是并行执行的，而每一个进程内部的语句是顺序执行的。

本节主要介绍赋值语句、IF 语句、CASE 语句、LOOP 语句。

一、顺序赋值语句

顺序语句中的赋值语句分变量赋值语句和信号赋值语句。其用途是将一个值或者表达式的运算结果传递给一个变量、信号或者由它们组成的数组。但二者是有区别的，变量赋值时间延迟为零，而信号赋值一定存在时间延迟。

1. 变量赋值语句

格式：目标变量名:=赋值源(表达式);

例如：x:=5.0;

z:=x+y;

其中 x 和 z 都是变量，:=表示给变量赋值。

2. 信号赋值语句

格式：目标信号名<=赋值源;

例如：y<=3;　　b(3 TO 4)<=c(1 TO 2);

其中 y 为信号，而 b 则为数组型信号，<=表示给信号赋值。

二、IF 语句

IF 语句的书写格式如下三种：

1. 简化的 IF 语句

```
IF　条件表达式　THEN
    顺序语句;
END　IF;
```

第一种语句的执行情况是，当程序执行到该语句时，首先检测关键词“IF”后的“条件表达式”的布尔值是否为真（TURE）。如果为真，那么 THEN 将顺序执行“顺序语句”中的各条语句，直到“END　IF;”；如果为假（FALSE），则不执行“顺序语句”，直接到跳到“END　IF”，结束 IF 语句的执行。

【例 2.3】用 IF 语句描述一个简单的 D 触发器。

```
ENTITY　dff1　IS
    PORT(d,clk:IN　BIT;
```

```
            q:OUT   BIT);
    END   dff1;
    ARCHITECTURE   one   OF   dff1   IS
    BEGIN
       PROCESS(clk)
          BEGIN
             IF   (clk'event   AND   clk='1')    THEN
               q<=d;
             END   IF;
        END   PROCESS;
    END   one;
```

2. 两分支结构

```
            IF   条件表达式   THEN
                 顺序语句;
            ELSE
                 顺序语句;
            END   IF;
```

两分支结构的执行情况是，当关键词“IF”后的“条件表达式”为真时，则顺序执行其下面的“顺序语句”中的各条语句，当“顺序语句”执行完成后，直接跳到“END IF;”，结束 IF 语句的执行；当“IF”后的“条件表达式”为假时，则程序直接跳到关键词“ELSE”，执行其下面的顺序语句，直到“END IF;”。因此它是一种两分支的结构。

【例 2.4】用 IF 语句描述一个 2 输入端的与门。

```
ENTITY   myand2   IS
        PORT(a,b:IN   BIT;
             y:OUT   BIT);
    END   myand2;
    ARCHITECTURE   one   OF   myand2   IS
    BEGIN
      PROCESS(a,b)
        BEGIN
          IF   (a='1'   AND   b='1')   THEN
            y<='1';
          ELSE
            y<='0';
          END   IF;
```

```
    END  PROCESS;
END  one;
```

3. 多分支结构

```
IF  条件表达式  THEN
  顺序语句;
ELSIF  条件表达式  THEN
  顺序语句;
 ⋮
ELSE
  顺序语句;
END  IF;
```

多分支结构通过关键词“ELSIF”设定了多个判断条件（条件表达式），这些判断条件的优先顺序（从高到低）与书写“条件表达式”的先后顺序一致。当某一判断条件为真时，则顺序执行其后的“顺序语句”，执行完成后，直接跳到“END IF;”当某一判断条件为假时，则跳过其后的“顺序语句”，再测试下一个条件表达式。当所有的条件表达式都为假时，则执行关键词“ELSE”下的“顺序语句”。因此它是一种多分支结构，功能与并行语句中的条件信号赋值语句类似。

【例 2.5】8 线-3 线优先编码器的设计

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY coder IS
PORT(a:IN STD_LOGIC_VECTOR(0 TO 7);
     y:OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END coder;
ARCHITECTURE rtl OF coder IS
BEGIN
PROCESS(a)
  BEGIN
    IF a(7)='0'          THEN y<="111";
    ELSIF (a(6)='0')     THEN y<="110";
    ELSIF (a(5)='0')     THEN y<="101";
    ELSIF (a(4)='0')     THEN y<="100";
    ELSIF (a(3)='0')     THEN y<="011";
    ELSIF (a(2)='0')     THEN y<="010";
    ELSIF (a(1)='0')     THEN y<="001";
```

```
        ELSE     y<="000";
         END IF;
    END PROCESS;
  END rtl;
```

三、CASE 语句

CASE 语句即选择语句，它根据表达式的值，从多项顺序语句中选择满足条件的一项来执行。CASE 语句也是 VHDL 的常用流程控制语句。CASE 语句的书写格式如下：

```
CASE 表达式 IS
When 选择值 =>顺序语句;
When 选择值 =>顺序语句;
          ⋮
When OTHERS =>顺序语句;
END CASE;
```

注意：“=>”不是运算符，相当“THEN”。

说明：

① 语句执行时，首先计算“表达式”的值，然后执行与“表达式”值相同的“选择值”后的“顺序语句”，最后“END CASE”。这里要注意，“=>”不是操作符，它只相当于“THEN”的作用。从这一点上来说，CASE 语句和并行选择信号赋值语句的功能相当。

② 条件句中的“选择值”必须在“表达式”的取值范围内，并且要完全覆盖“表达式”的所有取值。当“选择值”不能完全覆盖“表达式”的所有取值时，则最后一个条件语句中的“选择值”必须用“OTHERS”表示，以表示“选择值”未能列出“表达式”的其他取值。关键词“OTHERS”只能出现一次，且只能作为最后一种条件取值。这一点对于定义成 STD_LOGIC 和 STD_LOGIC_VECTOR 的数据类型尤为重要。

③“选择值”可以有以下四种不同的表达方式：单个数值，如 4；数值选择范围，如(2 TO 4)，表示取值为 2、3 和 4；并列数值，如 3|5，表示取值为 3 或 5；混合方式，以上三种方式的混合。

④ 每一条件语句中的“选择值”只能出现一次，不能有相同“选择值”的条件语句。

⑤ 与 IF 语句相比，CASE 语句组的程序可读性比较好，这是因为它把条件中“表达式”的所有取值都列了出来，可执行条件一目了然。但对相同的逻辑功能描述，综合后，用 CASE 语句描述比 IF 描述耗用更多的硬件资源。因此，对本身就具有优先逻辑关系的描述（如优先编码器），采用 IF 语句比 CASE 语句更

合适。

【例 2.6】用 CASE 语句描述 4 选 1 数据选择器。

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
ENTITY  mux4_1  IS
        PORT(a,b,c,d:IN  STD_LOGIC;
        sel:IN  STD_LOGIC_VECTOR(1  DOWNTO  0);
        y:OUT  STD_LOGIC);
END  mux4_1;
ARCHITECTURE  one  OF  mux4_1  IS
BEGIN
   PROCESS(sel,a,b,c,d)
      BEGIN
        CASE  sel  IS
           WHEN  "00"=>y<=a;
           WHEN  "01"=>y<=b;
           WHEN  "10"=>y<=c;
           WHEN  "11"=>y<=d;
           WHEN  OTHERS=>y<='X';
         END  CASE;
   END  PROCESS;
END  one;
```

四、LOOP 语句

LOOP 语句就是循环语句，它可以使所包含的一组顺序语句被循环执行。

```
[标号:]FOR  循环变量  IN  循环范围  LOOP
            顺序语句;
        END  LOOP  [标号];
```

常用的书写格式如下：

①“循环变量”是一个临时变量，属于 LOOP 语句的局部变量，它由 LOOP 语句自动定义，不必事先定义。该变量只能作为赋值源，不能被赋值。使用时应注意，在 LOOP 语句范围内不要再使用其他与此循环变量同名的标识符。

②“循环范围”规定循环语句被执行的次数。“循环变量”从“循环范围”的初值开始，每执行完一次“顺序语句”后递增 1，直到达到“循环范围”指定的最大值。

【例 2.7】用 FOR_LOOP 语句描述图 2–3 的电路。

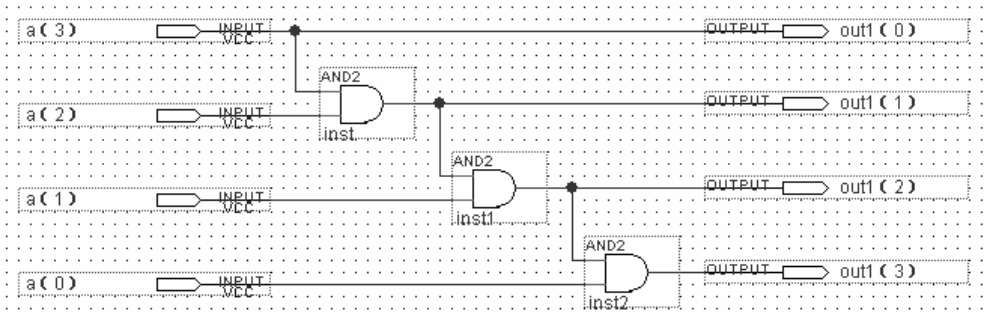


图 2–3　例 2.7 图

```
ENTITY  exam_1  IS
        PORT(a:IN  BIT_VECTOR(0  TO  3);
        out1:OUT  BIT_VECTOR(0  TO  3));
END  exam_1;
ARCHITECTURE  beh  OF  exam_1  IS
    BEGIN
    PROCESS(a)
    VARIABLE  b:BIT;
BEGIN
         b:='1';
         FOR  i  IN  0  TO  3  LOOP
         b:=a(3-i)  AND  b;
         out1(i)<=b;
         END  LOOP;
       END  PROCESS;
    END  beh;
```

2.3　VHDL 程序的并行语句

并行语句结构是最具 VHDL 特色的，是与一般软件程序最大的区别所在。在 VHDL 中，并行语句有多种语句格式，各种并行语句的执行都是同步进行的，或者说是并行运行的，其执行的方式与书写的顺序无关。这种并行性是由硬件本身的并行性决定的，即一旦电路接通电源，它的各部分就会按照事先设计好的方案同时工作。并行语句在执行时，各并行语句之间可以有信息来往，也可以互为独立、互不相关。另外，每一并行语句内部的语句可以有两种不同的运行方式，即并行执行方式（如块语句）和顺序执行方式（如进程语句）。

本节介绍结构体中常用的并行语句：进程（PROCESS）语句、并行信号赋值语句、元件例化语句。

一、进程（PROCESS）语句

进程语句是 VHDL 设计中用得最多的语句之一，进程语句本身是并行语句，其内部却是由顺序语句组成。一个结构体内可以包含多个进程语句，多个进程之间是并发执行的，多个进程语句间可以通过信号来交换信息。进程语句的格式有两种。

格式一：

```
[进程标号:]  PROCESS  [(敏感信号参数表)]  [IS]
  [进程声明部分;]              --说明用于该进程的常量，变量和子程序
  BEGIN
      顺序描述语句;
END  PROCESS  [进程标号];
```

此格式有敏感信号参数表，其内部不允许存在 WAIT 语句。

格式二：

```
[进程标号:]  PROCESS  [IS]
  [进程声明部分;]              --说明用于该进程的常量，变量和子程序
  BEGIN
  WAIT 语句;
  顺序描述语句;
END  PROCESS  [进程标号];
```

进程语句是最常用的并行语句，既可以用来描述组合逻辑电路，又可以描述时序逻辑电路，是 VHDL 程序设计中应用最多也是最能体现硬件描述语言特点的一种语句。进程语句的主要特点归纳如下：

① 同一结构中的各个进程之间是并发执行的，并且都可以使用实体说明和结构体中所定义的信号；而同一进程内部的语句则必须是顺序语句。

② 为启动进程，进程语句中必须至少包含一个敏感信号或包含一个 WAIT 语句，但是在一个进程中不能同时存在敏感信号和 WAIT 语句。

③ 一个结构体中的各个进程之间可以通过信号来进行通信，进程的声明部分不能定义信号，但可定义进程的内部变量。

④ 在进程内部信号赋值是有时间延迟的，在同一个进程中对同一个信号多次赋值时，起作用的是最后一次条件成立的赋值；而在进程内部变量赋值是无时间

延迟的，在同一个进程中对同一个变量多次赋值时，每次都立即起作用。

【例 2.8】一个典型的结构体实例。

```
ARCHITECTURE  a  OF  states_mach  IS
BEGIN
  P1:PROCESS(clk)
     BEGIN
        IF (clk'event  AND  clk='1')  THEN
          current_state<=next_state;
        END  IF;
  END  PROCESS;
  ⋮
END  A;
```

上例中，P1 为进程标号，时钟 clk 为敏感信号。每当 clk 发生一次变化时，BEGIN 和 END PROCESS 之间的顺序语句就会运行一次。由于时钟 clk 变化包括上升沿和下降沿，为了准确描述，在顺序语句中用了一个条件判断语句“IF (clk'event AND clk='1') THEN”来判断 clk 的上升沿。若要判断 clk 的下降沿，可用“IF (clk'event AND clk='0') THEN”语句。

二、并行信号赋值语句

并行信号赋值语句有三种形式：简单信号赋值语句、选择信号赋值语句和条件信号赋值语句。这三种信号赋值语句的共同特点是赋值目标必须都是信号，所有赋值语句与其他并行语句一样，在结构体中的执行是同时发生的。

1. 简单信号赋值语句

简单信号赋值语句是并行语句结构的最基本单元，它的语句书写格式如下：

赋值目标<=表达式;

应用该赋值语句时一定要注意，赋值目标的数据对象必须是信号，赋值运算符“<=”两边的数据类型必须一致。

【例 2.9】本例的结构体中描述了 3 个基本的逻辑门。

```
ENTITY  gate  IS
  PORT(a,b:IN  BIT;
       y1,y2,y3:OUT  BIT);
  END  gate;
ARCHITECTURE  one  OF  gate  IS
BEGIN
   y1<=a  AND  b; --与门
   y2<=a  OR  b;     --或门
```

```
        y3<=NOT  a;        --非门
    END  one;
```

2. 选择信号赋值语句

选择信号赋值语句的书写格式如下：

```
WITH  选择表达式  SELECT
    赋值目标信号<=表达式  WHEN  选择值,
                  表达式  WHEN  选择值,
                   ⋮
                  表达式  WHEN  选择值;
```

① 当“选择表达式”的值发生变化时，将启动此语句对各子句的“选择值”进行测试对比。当发现有满足条件的子句时，就将此子句中的“表达式”值赋给“赋值目标信号”。

② 每条子句应以“,”结束，最后一条子句以“;”结束。

③“选择值”不能有重复，且“选择值”应包含“选择表达式”的所有取值，不允许存在选择值涵盖不全的情况。

【例 2.10】用选择信号赋值语句设计一个 4 选 1 数据选择器，其符号如图 2–4 所示。其中 SEL[1..0]为 2 位地址输入端，A、B、C、D 为数据输入端，Y 为数据输出端。当 SEL[1..0]为“00”时，Y=A；当 SEL[1..0]为“01”时，Y=B；当 SEL[1..0]为“10”时，Y=C；当 SEL[1..0]为“11”时，Y=D。

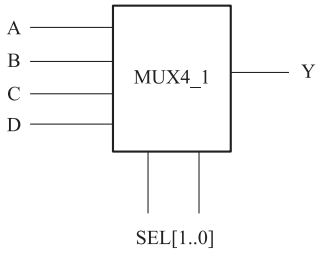


图 2–4　例 2.10 图

```
ENTITY  mux4_1  IS
    PORT(a,b,c,d:IN  BIT;
         sel:IN  BIT_VECTOR(1  DOWNTO  0);
         y:OUT  BIT);
END  mux4_1;
ARCHITECTURE  one  OF  mux4_1  IS
BEGIN
    WITH  sel  SELECT
        y<=a  WHEN  "00",
           b  WHEN  "01",
           c  WHEN  "10",
           d  WHEN  "11";
END  one;
```

3. 条件信号赋值语句

条件信号赋值语句书写格式如下：

```
目标信号<=表达式  WHEN  赋值条件  ELSE
          表达式  WHEN  赋值条件  ELSE
            ⋮
          表达式;
```

① 条件信号赋值语句与选择信号赋值语句的最大区别在于后者的各个“选择值”之间处于同一优先级，而前者的各个“赋值条件”具有优先顺序，优先级由高到低的顺序与语句书写顺序一致。

② 当某个“赋值条件”得到满足（即其值为“真”）时，立即将该条件“WHEN”前的“表达式”值赋给“目标信号”；当几个“赋值条件”都得到满足时，优先级高的那个条件“WHEN”前的“表达式”值赋给“目标信号”；当所有的“赋值条件”都得不到满足时，最后一个“ELSE”关键词后的“表达式”值赋给“目标信号”。

③ 每行语句后没有标点符号，最后一行“表达式”用“;”结束。

【例 2.11】用条件信号赋值语句设计 4 选 1 数据选择器。

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
ENTITY  mux4_1  IS
   PORT(a,b,c,d:IN  BIT;
        sel:IN  BIT_VECTOR(1  DOWNTO  0);
        y:OUT  BIT);
END  mux4_1;
ARCHITECTURE  one  OF  mux4_1  IS
BEGIN
   y<=a  WHEN  sel="00"  ELSE
     b  WHEN  sel="01"  ELSE
     c  WHEN  sel="10"  ELSE
     d;
END  one;
```

三、元件例化语句

元件例化语句引入的是一种连接关系，即将预先设计好的设计实体定义成一个元件，然后利用例化语句将此元件与当前的设计实体中的指定端口相连接，从而为当前设计实体引入了一个低一级的设计层次。也可以这样来理解例化语句：当前设计实体相当于一个较大的电路系统，预先设计好的设计实体相当于一个要插在这个电路系统板上的芯片，而当前设计实体中的例化语句则相当于这块电路板上准备接收此芯片的一个插座。

元件例化语句通常由元件声明和元件例化两部分组成。语句书写格式如下：

```
--元件声明部分
  COMPONENT  元件名
    GENERIC(参数表);
    PORT(端口信息);
    END  COMPONENT;
--元件例化部分
   例化名:元件名  PORT  MAP(端口名=>连接端口名,…);
```

① 第一部分的“元件声明”，是对预先设计好的元件的定义语句，相当于对一个已有的设计实体进行封装，使其只留出对外的接口界面，也就像一个集成芯片只对外留出几个引脚一样。“类属表”可定义一些参数；“端口信息”可列出已有元件端口的名称、模式、数据类型。该部分可放在结构体中“语句说明部分”。

② 第二部分的“元件例化”，用于说明当前设计实体和被调用元件的连接关系。其中，“例化名”是必须的，它类似于当前电路系统板上的一个插座名；而“元件名”则是已定义好的、准备在此插座上插入的元件（或芯片）名称；“端口名”是已有的元件（或芯片）的端口名称；“=>”是关联（连接）符号；“连接端口名”则是当前系统与准备接入的元件对应端口相连的通信端口，相当于插座上各插针的引脚名。

③ 已有元件的“端口名”与当前系统的“连接端口名”的关联描述有两种方式。一种是上述方式，在这种方式下，已有元件的“端口名”、关联符号“=>”都是必须存在的，“端口名=>连接端口名”在 PORT MAP 语句中的位置可以是任意的；另一种是位置关联方式，在这种方式下，“端口名”和“=>”都可省去，在 PORT MAP 子句中只列出当前系统中的“连接端口名”即可，但要求“连接端口名”的排列顺序与“元件声明”中已有元件“端口信息”中所列的端口名排列顺序相同，书写格式为

```
   例化名:元件名  PORT  MAP(连接端口名 1，连接端口名 2，…);
```

【例 2.12】利用例化语句设计图 2–5 所示的电路。

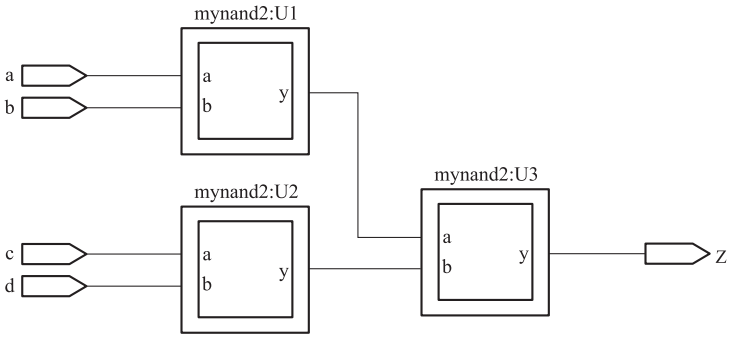


图 2–5 例 2.12 图

```
--首先完成与非门的设计
    LIBRARY  IEEE;
    USE  IEEE.STD_LOGIC_1164.ALL;
    ENTITY  mynand2  IS
      PORT(a,b:IN  STD_LOGIC;
           y:OUT  STD_LOGIC);
    END  mynand2;
ARCHITECTURE  one  OF  mynand2  IS
    BEGIN
      y<=a  NAND  b;
    END  one;
--利用例化语句完成设计
    LIBRARY  IEEE;
    USE  IEEE.STD_LOGIC_1164.ALL;
    ENTITY  ord4  IS
      PORT(a,b,c,d:IN  STD_LOGIC;
           z:OUT  STD_LOGIC);
    END  ord4;
        ARCHITECTURE  behv  OF  ord4  IS
    COMPONENT  mynand2      --元件声明部分
      PORT(a,b:IN  STD_LOGIC;
           y:OUT  STD_LOGIC);
    END  COMPONENT;
SIGNAL  temp1,temp2:STD_LOGIC;
    BEGIN
                                        --元件例化部分
      U1: mynand2  PORT  MAP(a,b,temp1); --按位置关联
      U2: mynand2  PORT  MAP(a=>c,b=>d,y=>temp2);
                                          --按名字关联
      U3: mynand2  PORT  MAP(temp1,temp2,y=>z);
                                          --混合关联
    END  behv;
```

四、生成语句（GENERATE）

生成语句具有复制作用，它可以生成与已有的某个元件或设计单元电路完全相同的一组并行元件或设计单元电路结构。

生成语句的书写格式有以下两种形式：

形式一：

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

形式二：

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

① 生成语句的生成方式有“FOR”和“IF”两种，它们用于规定并行语句的复制方式。

②“说明语句”用于对元件的数据类型、数据对象等作一些局部说明。

③“并行语句”是用来复制的基本单元，主要包括元件、进程语句、块语句、并行信号赋值语句，甚至生成语句，这表示生成语句允许存在嵌套结构，因而可用于生成元件的多维阵列结构。

④“标号”并非是必需的，但如果在嵌套式生成语句结构中就是十分重要的。

⑤ 对于“FOR”语句结构，主要用来描述设计中的一些有规律的单元结构，其“循环变量”是一个局部变量，它根据“取值范围”而自动递增或递减。“取值范围”的书写格式有以下两种方式。

```
表达式  TO  表达式;                    --递增方式，如 1 TO 5
表达式  DOWNTO  表达式;                --递减方式，如 5 DOWNTO 1
```

【例 2.13】利用 GENERATE 语句产生 4 个 D 触发器，如图 2–6 所示。

```
LIBRARY  IEEE;
    USE  IEEE.STD_LOGIC_1164.ALL;
    ENTITY  dff_4  IS
        PORT(clk,clrn,prn:IN  STD_LOGIC;
             d:IN  STD_LOGIC_VECTOR(3  DOWNTO  0);
             q:OUT  STD_LOGIC_VECTOR(3  DOWNTO  0));
    END  dff_4;
    ARCHITECTURE  beh  OF  dff_4  IS
      COMPONENT  dff                          ----元件声明
        PORT(d,clk,clrn,prn:IN  STD_LOGIC;
             q:OUT  STD_LOGIC);
```

```
 END    COMPONENT;
      BEGIN
d4:FOR i IN 3 DOWNTO 0 GENERATE    --生成语句
    u:dff PORT MAP(d(i),clk,clrn,prn,q(i));
                                          --元件例化
END    GENERATE;       END    beh;
```

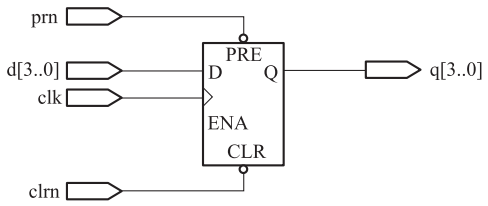


图 2–6　例 2.13 图

2.4　VHDL 程序的语言要素总结

VHDL 的语言要素，作为硬件描述语言的基本结构元素，主要有数据对象、数据类型和运算操作符，而数据对象则包括变量、信号和常数三种。

一、文字语法

VHDL 文字主要包括数字文字、字符串文字、标识符和下标名。

1. 数字文字

数字文字有多种表达方式，举例如下。

（1）整数文字

整数文字都是十进制数，例如

3，235，0，178E2(=17800)，45_146(=45146)

数字间的下画线只是为了提高文字的可读性，相当于一个空的间隔符，不影响数值本身的意义。

（2）实数文字

实数文字是十进制数，必须带有小数点，例如

123.57，45_146.571，0.0，3.15

（3）以数制基数表示的文字

由五个组成部分。

第一部分，用十进制数标明数制进位的基数；

第二部分，数制隔离符号“#”；

第三部分，表达的文字；

第四部分，指数隔离符号“#”；

第五部分，用十进制表示的指数部分，如果是 0 可以省略。例如

10#150#　　--(十进制数表示，=150)

2. 字符串文字

字符是用单引号引起来的 ASCII 字符，可以是数值，也可以是符号或字母，如：'A'、'X'、'*'、'R'。而字符串是一维的数组，必须放在双引号中。VHDL 有两种类型的字符串：文字字符串和数位字符串。

（1）文字字符串

文字字符串是用双引号引起来的一串文字，例如

"ERROR"、"BB$CC"、"BOTH S AND Q EQUAL TOL"

（2）数位字符串

数位字符串是二进制、八进制或十六进制的数组，首先要有计算基数，然后将该基数表示的值放在双引号中，基数符号以"B"、"O"和"X"表示，并放在字符串的前面。

B：二进制基数符号，表示二进制数位 0 或 1，表示一个 bit。

O：八进制基数符号，代表一个 3 位的二进制数。

X：十六进制基数符号，代表一个 4 位的二进制数。例如

B"1_1101_1110"　　--二进制数数组，数组长度的 9

X"AD0"　　--十六进制数数组，数组长度是 12

3. 标识符

标识符用来定义常数、变量、信号、端口、子程序或参数的名字。VHDL 的基本标识符就是以英文字母开头，不连续使用下画线“_”，不以下画线“_”结尾，由 26 个大小写英文字母、数字以及下画线组成的字符串。例如：

DECODER_1，FFT，Sig_N，NOT_ACK 是合法的标识符；

_DECODER_1，2FFT，Sig_#N，NOT-ACK 是非法标识符。

4. 下标名

下标名用于指示数组型变量或信号的某一元素，其语句格式如下：

表达式 1　TO/DOWNTO　表达式 2

表达式的数值必须在数组元素下标号范围内，并且必须是可计算的。TO 表示数组下标序列由低到高，如“2 TO 8”；DOWNTO 表示数组下标序列由高到低，如“8 DOWNTO 2”。

二、数据对象

在 VHDL 中，数据对象类似于一种容器，它接受不同数据类型的赋值。数据对象有三种，即变量（VARIABLE）、常量（CONSTANT）和信号（SIGNAL）。

1. 变量（VARIABLE）

在 VHDL 中，变量是一个局部量，只能在进程和子程序中使用。变量不能将信息带出对它作出定义的当前设计单元。变量的赋值是一种理想化的数据传输，是立即发生，不存在任何延时行为。定义变量的语法格式如下：

VARIABLE 变量名:数据类型:=初始值;

例如：

```
VARIABLE   A:INTEGER;            --定义 A 为整数型变量
VARIABLE   B,C:INTEGER:=2;       --定义 B 和 C 为整型变量，初值为 2
```

2. 信号（SIGNAL）

信号可看做是硬件连线的一种抽象表示，它既能保持变化的数据，又可连接各元件作为元件之间数据传输的通路。信号通常在结构体、程序包和实体中说明。信号的语法格式如下：

SIGNAL 信号名:数据类型:=初始值

例如：

```
SIGNAL   S1:STD_LOGIC:=0;        --定义了一个标准的单值信号 S1，初始
                                   值为低电平
```

信号是一个全局变量，可以用来进行进程之间的通信，在 VHDL 语言中对信号赋值一般是按仿真时间来进行的，而且信号值的改变也需按仿真时间的计划行事。

3. 常量（CONSTANT）

常量是一个固定的值。所谓常量说明，就是对某一常数名赋予一个固定的值。通常赋值在程序开始前进行，该值的数据类型则在说明语句中指明。常量的语法格式如下：

CONSTANT 常量名:数据类型:=表达式

例如：

```
CONSTANT   VCC:REAL:=5.0;
```

常量定义的语句所允许的设计单元有实体、结构体、程序包、块、进程和子程序。在程序包中定义的常量可以暂时不设具体数值，它可以在程序包体中设定。

三、数据类型

在 VHDL 语言中，每个客体都有特定的数据类型。为了能够描述各种硬件电路，创建高层次的系统和算法模型，VHDL 具有丰富的数据类型，除了有很多预定义的数据类型可直接使用外，用户还可自定义数据类型，这给设计人员带来了很大的自由和方便。

1. 标准的数据类型

（1）位（BIT）

在数字系统中，信号通常用一个位来表示。位值的表示方法是，用字符 0 或

1 放在单引号中表示。位和整数中的 0 和 1 不同，'0'和'1'仅仅表示一个位的两种取值。

（2）位矢量（BIT_VECTOR）

位矢量是用双引号括起来的一组位数据。例如："001100"，H"00BE"。位矢量前的 H 表示是十六进制。

（3）布尔量（BOOLEAN）

布尔数据类型实际上是一个二值枚举型数据类型，它的取值有 FALSE 和 TRUE 两种。例如，在 IF 语句中，测试结果产生一个布尔量 FALSE 或 TRUE，并以此结果控制其他语句的执行与否。如语句“IF clk='1'THEN …”在信号 clk 确实为“1”的情况下，表达式“clk='1'”取值为 TRUE，此时将执行 THEN 后的语句，否则 THEN 后的语句不会被执行。

（4）整数（INTEGER）

32 位有符号的二进制数。

（5）实数（REAL）

与数学上的实数定义相同，也称为浮点数，仅能在 VHDL 仿真器中使用。

（6）字符（CHARACTER）和字符串（STRING）

字符数据类型使用单引号，字符串使用双引号标明，如: 'A'、"abcd"。

（7）时间（TIME）

VHDL 中唯一的预定义物理类型是时间。

（8）错误等级（SEVERITY_LEVEL）

指示 VHDL 在编译、综合、仿真过程中的工作状态，共有四种可能的状态：NOTE（注意）、WARNING（警告）、ERROR（出错）、FAILURE（失败）。

2. IEEE 预定义的标准逻辑位与矢量

STD_LOGIC 在 IEEE 库的程序包 STD_LOGIC_1164 中的定义格式如下：

TYPE STD_LOGIC IS ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');

STD_LOGIC _VECTOR 数据类型在 IEEE 库的程序包 STD_LOGIC_1164 中的定义格式如下：

TYPE STD_LOGIC_VECTOR IS ARRAY (NATURAL RANGE<>) OF STD_LOGIC;

使用时须加入下面的库语句：

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

3. 自定义数据类型

（1）枚举类型（ENUMERATED）

这是用文字符号代替一组实际的二进制数的特殊数据类型，如，将一个星期 WEEK 定义为 7 个状态的枚举数据类型：

TYPE WEEK IS (sun, mon, tue, wed, thu, fri, sat);

（2）整数类型（INTEGER）和实数类型（REAL）

对已作过预定义的数据类型，做取值范围约束，如：

TYPE nat IS INTEGER RANGE 0 TO 255;

--定义数 nat 的取值范围为 0～255

（3）数组类型（ARRAY）

可以定义约束型数组和未约束型数组。约束型数组定义格式如下:

TYPE 数组名 IS ARRAY (数组下标范围) OF 数据类型;

如：

TYPE BIT_VECTOR IS ARRAY (NATURAL RANGE<>)OF BIT;

VARABLE VA:BIT_VECTOR(1 TO 6); --表示将数组取值范围定在 1~6

其中，符号“<>”是下标范围待定符号，用到数组类型时，再填入具体的数值范围。

（4）用户自定义类型

VHDL 允许用户自行定义新的数据类型，用户自定义数据类型是用类型定义语句 TYPE 和子类型语句 SUBTYPE 实现的。子类型定义的一般格式为

SUBTYPE 子类型名 IS 数据类型名[范围]

例如：在 STD_LOGIC_VECTOR 基础上形成的子类型：

SUBTYPE iobus IS STD_LOGIC_VECTOR(4 DOWNTO 0)

子类型可以通过对原数据类型指定范围而形成，也可以完全和原数据类型范围一致。子类型常用于存储器阵列的数组描述的场合。

四、运算操作符

1. 算术操作符

求和操作符：包括加法操作符、减法操作符和并置操作符。

符号操作符：包括“＋”（正）和“－”（负）两种操作符。

求积操作符：包括*（乘）、/（除）、MOD（取模）和 RED（取余）四种。

混合操作符：包括乘方“**”和取绝对值“ABS”两种。

移位操作符：包括 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）和 ROR（逻辑循环右移）6 种操作符。

2. 关系操作符

VHDL 提供了 6 种关系操作符，其中“=”和“/=”用于数值比较，“>”、“<”、“>=”和“<=”用于关系排序判断。

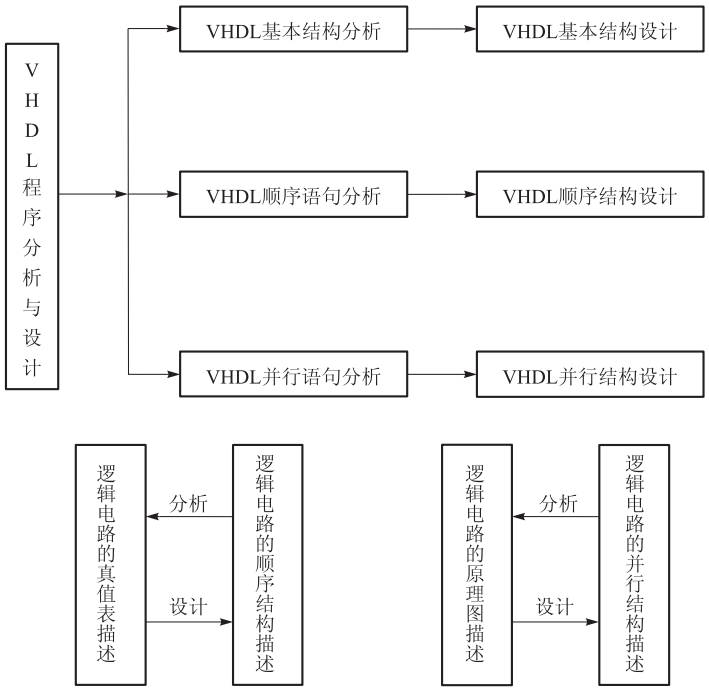
3. 逻辑操作符

VHDL 提供了 7 种基本逻辑操作符，它们是 AND、OR、NOT、NAND、NOR、XOR 和 XNOR。

4. 重载操作符

VHDL 中的重载操作符是为了使不同数据类型的数据对象之间能够进行运算，对原有的基本操作符重新作了定义，构成的新的操作符。

本章小结



本章习题

1. 根据下面的 VHDL 语句，描述出相应的电路原理图。

```
 LIBRARY    ieee;
 USE        ieee.std_logic_1164.ALL;
 USE        ieee.std_logic_unsigned.ALL;
 ENTITY cfq_1      IS
  PORT    (d,cp      :IN   std_logic;
           q,nq      :OUT std_logic);
 END      cfq_1;
 END      ar_4;
ARCHITECTURE    ar_4    OF     cfq_1 IS
```

```
BEGIN
 PROCESS (CP)
 BEGIN
      IF    cp='1'    THEN
            q <= d;
            nq <=NOT d;
      END    IF;
 END    PROCESS;
```

2. 下面代码中的条件信号赋值语句无 ELSE 部分，正确吗？上机编辑输入、编译、仿真下面代码；通过编译、仿真、执行 TOOLS→RTL Viewer 命令，打开 RTL 电路观察器观察此电路的 RTL 原理图，并解释代码描述的是什么电路。

```
ENTITY dtff IS
   GENERIC(initial:BIT:= '1');
   PORT(d,clock:IN BIT;q:BUFFER BIT:=initial);
END dtff;
ARCHITECTURE a OF dtff IS
   BEGIN
   q<=d WHEN (clock'EVENT AND clock='1');
END a;
```

3. 分析下面 VHDL 代码，写出其描述电路得真值表，说明其功能。

```
ENTITY maj_c IS
   PORT(a,b,c:IN BIT;m:OUT BIT);
END maj_c;
ARCHITECTURE concurrent OF maj_c IS
BEGIN
       WITH a&b&c SELECT
         m<='1'WHEN"110"｜"101"｜"011"｜"111", '0' WHEN OTHERS;
- -"110"｜"101"｜"011"｜"111"表示取值"110"或"101"或"011"或"111"
END concurrent;
```

4. 分析下面 VHDL 代码，写出其描述电路的真值表，说明其功能；上机编辑输入、输出、编译、仿真下面代码，执行 Tools→RTL Viewer 命令，打开 RTL 电路观察器观察此电路的 RTL 原理图。

```
ENTITY halfsub IS
   PORT(A,B:IN BIT;
          S,C:OUT BIT);
END halfsub:
```

```
ARCHITECTURE a OF halfsyb IS
BEGIN
   PROCESS(A,B)
   BEGIN
      S<=A XOR B AFTER 10 ns
      C<= (NOT A) AND B AFTER 10 ns;
   END PROCESS;
EBD a;
ENTITY orgate IS
   PORT(A1,B1:IN BIT;
            O1 :OUT BIT);
END orgate;
ARCHITECTURE a OF orgate IS
BEGIN
   O1<=A1 OR B1;
END a;
ENTITY fullsub IS
   PORT(I1,I2,C_IN:IN BIT;
                  FS,C_OUT:OUT BIT);
END fullsub;
ARCHITECTURE a OF fullsub IS
    SIGNAL temp_S,temp_c1,temp_2:BIT;
    COMPONENT halfsub
        PORT(A,B:IN BIT;
                   S,C:OUT BIT);
    END COMPONENT;
    COMPONENT orgate
        PORT(A1,B1:IN BIT;    01:OUT BIT);
    END COMPONENT;
BEGIN
   U0:halfsub PORT MAP(I1,I2,temp_S,temp_c1);
   U1:halfsub PORT MAP(temp_S,C_IN,FS,temp_c2);
   U2:orgate PORT MAP(temp_c1,temp_c2,C_OUT);
END a;
```

5. 下面是描述七段译码器的 VHDL 代码，应用 Quartus Ⅱ的 HDL 输入法，输入法编辑下面代码，并通过编译和仿真。

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY decl7s IS
   PORT(d :IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         Led:OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END;
ARCHITECTURE a OF DECL7S IS
BEGIN
   PROCESS(d)
   BEGIN
   CASE d IS
      WHEN"0000"=> led<="0111111";
      WHEN"0001"=> led<="0000110";
      WHEN"0010"=> led<="1011011";
      WHEN"0011"=> led<="1001111";
      WHEN"0100"=> led<="1100110";
      WHEN"0101"=> led<="1101101";
      WHEN"0110"=> led<="1111101";
      WHEN"0111"=> led<="0000111";
      WHEN"1000"=> led<="1111111";
      WHEN"1001"=> led<="1101111";
      WHEN"1010"=> led<="1110111";
      WHEN"1011"=> led<="1111100";
      WHEN"1100"=> led<="0111001";
      WHEN"1101"=> led<="1011110";
      WHEN"1110"=> led<="1111001";
      WHEN"1111"=> led<="1110001";
      WHEN OTHERS=> NULL;
      END CASE;
   END PROCESS;
END a;
```