第 3 章

VHDL 硬件描述语言

3.1 VHDL 简介

VHDL 是一种用于电路设计的高级语言，20 世纪 80 年代后期出现，最初是由美国国防部开发出来供美军用来提高设计的可靠性和缩减开发周期的一种使用范围较小的设计语言。但是，由于它在一定程度上满足了当时的设计需求，于是在 1987 年成为 IEEE 的标准(IEEE STD 1076—1987)。1993 年更进一步修订后，变得更加完备，成为 IEEE 的 IEEE STD 1076—1993 标准。目前，大多数的 CAD 厂商出品的 EDA 软件都兼容了这种标准。VHDL 的英文全写是 VHSIC(Very High Speed Integrated Circuit) Hardware Description Language，翻译成中文就是超高速集成电路硬件描述语言，因此它主要应用在数字电路的设计中。

常用硬件描述语言有 VHDL、Verilog 和 ABEL。VHDL 起源于美国国防部的 VHSIC，Verilog 起源于集成电路的设计，ABEL 则来源于可编程逻辑器件的设计。下面从使用方面将三者进行对比。

(1) 逻辑描述层次：一般的硬件描述语言可以在 3 个层次上进行电路描述，其层次由高到低依次可分为行为级、RTL 级和门电路级。VHDL 是一种高级描述语言，适用于行为级和 RTL 级的描述，最适于描述电路的行为；Verilog 和 ABEL 是较低级的描述语言，适用于 RTL 级和门电路级的描述，最适于描述门级电路。

(2) 设计要求：VHDL 进行电子系统设计时可以不了解电路的结构细节，设计者所做的工作较少；Verilog 和 ABEL 进行电子系统设计时需了解电路的结构细节，设计者需做大量的工作。

(3) 综合过程：任何一种语言源程序，最终都要转换成门电路级才能被布线器或适配器所接受。因此，VHDL 源程序的综合通常要经过行为级→RTL 级→门电路级的转化，VHDL 几乎不能直接控制门电路的生成。而 Verilog 和 ABEL 源程序的综合过程要稍简单，即经过 RTL 级→门电路级的转化，易于控制电路资源。

(4) 对综合器的要求：VHDL 描述语言层次较高，不易控制底层电路，因而对综合器的性能要求较高，Verilog 和 ABEL 对综合器的性能要求较低。

(5) 支持的 EDA 工具：支持 VHDL 和 Verilog 的 EDA 工具很多，但支持 ABEL 的

综合器仅仅 Dataio 一家。

(6) 国际化程度：VHDL 和 Verilog 已成为 IEEE 标准，而 ABEL 正朝国际化标准努力。

3.1.1 VHDL 的发展及特点

VHDL 是一种用普通文本形式设计数字系统的硬件描述语言，主要用于描述数字系统的结构、行为、功能和接口，可以在任何文字处理软件环境中编辑。除了含有许多具有硬件特征的语句外，其形式、描述风格及语法十分类似于计算机高级语言。VHDL 程序将一项工程设计项目(或称设计实体)分成描述外部端口信号的可视部分和描述端口信号之间逻辑关系的内部不可视部分，这种将设计项目分成内、外两个部分的概念是硬件描述语言(HDL)的基本特征。当一个设计项目定义了外部界面(端口)，在其内部设计完成后，其他的设计就可以利用外部端口直接调用这个项目。VHDL 的主要特点如下。

(1) 作为 HDL 的第一个国际标准，VHDL 具有很强的可移植性。

(2) 具有丰富的模拟仿真语句和库函数，随时可对设计进行仿真模拟，因而能将设计中的错误消除在电路系统装配之前，在设计早期就能检查设计系统功能的可行性，有很强的预测能力。

(3) VHDL 有良好的可读性，接近高级语言，容易理解。

(4) 系统设计与硬件结构无关，方便了工艺的转换，也不会因工艺变化而使描述过时。

(5) 支持模块化设计，可将大规模设计项目分解成若干个小项目，还可以把已有的设计项目作为一个模块调用。

(6) 对于用 VHDL 完成的一个确定设计，可以利用 EDA 工具进行逻辑综合和优化，并能自动地把 VHDL 描述转变成门电路级网表文件。

(7) 设计灵活，修改方便，同时也便于设计结果的交流、保存和重用，产品开发速度快，成本低。

3.1.2 传统设计与 VHDL 设计对照

1. 传统设计

传统的硬件设计方法有如下特征。

(1) 采用自下而上的设计方法。使用该方法进行硬件设计从选择元器件开始，并用这些元器件进行逻辑电路的设计，从而完成系统的硬件设计，然后将各功能模块连接起来，完成整个系统的硬件设计。

(2) 采用通用逻辑元器件。传统的设计方法通常采用 74 系列和 CMOS4000 系列的产品进行设计。

(3) 在系统硬件设计的后期进行调试和仿真。只有在部分或全部硬件电路设计连接完成后，才可以进行电路调试。一旦考虑不周，系统设计存在较大缺陷，则要重新设计，使得设计周期延长。

(4) 设计结果是一张电路原理图。当设计调试完成后，就会形成一张电路原理图，它包括元器件型号及信号之间的连接关系等。

传统的硬件设计方法已经使用了几十年，是广大电子工程师熟练掌握的一种方法，但它至今没有任何标准规范。因此，其设计效率低，系统性差，开发成本高，市场竞争力小。

2. VHDL 设计

VHDL 设计主要包括以下几个步骤。

（1）文本编辑。用任何文本编辑器都可以进行，也可以用专用的 HDL 编辑环境。通常 VHDL 文件保存为.vhd 文件，Verilog 文件保存为.v 文件。

（2）文件编译。使用编译工具编译源文件。

（3）功能仿真。将文件调入 HDL 仿真软件进行功能仿真，检查逻辑功能是否正确（也叫前仿真，对简单的设计可以跳过这一步，只在布线完成以后，进行时序仿真）。

（4）逻辑综合。将源文件调入逻辑综合软件进行综合，即把语言综合成最简的布尔表达式。逻辑综合软件会生成.edf 或.edif 的 EDA 工业标准文件。

（5）布局布线。将.edf 文件调入 PLD 厂家提供的软件中进行布线，即把设计好的逻辑安放在 CPLD/FPGA 内。

（6）时序仿真。需要利用在布局布线中获得的精确参数，用仿真软件验证电路的时序（也叫后仿真）。通常以上过程可以都在 CPLD/FPGA 厂家提供的开发工具中进行。

3.2　VHDL 程序的基本结构

3.2.1　VHDL 程序的基本单元与构成

本小节将通过一个比较典型的设计示例，使读者能迅速地从整体上把握 VHDL 程序的基本结构和设计特点，达到快速入门的目的，为以后各章的学习提供一个良好的开端。

图 3-1 是一个 2 选 1 的多路选择器逻辑框图，a 和 b 分别是两个数据输入端的端口名，s 为通道选择控制信号输入端的端口名，y 为输出端的端口名。其逻辑功能可表述为：若 s=0 则 y=a；若 s=1 则 y=b。

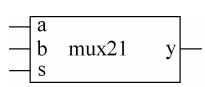

图 3-1　2 选 1 选择器 mux21

此选择器的功能可用如下的 VHDL 描述。

【程序 3-1】 2 选 1 选择器。

```
library ieee;
use ieee.std_logic_1164.all;                    } ieee 库使用说明
entity mux21 is
  port ( a,b : in std_logic;                    }
         s : in std_logic;                      } 器件 mux21 的外部接口信号说明，port 相当于器件的引脚说明，这一部分称为实体
         y : out std_logic );                   }
end entity mux21;
architecture one of mux21 is                    }
  begin                                         }
    y <= a when s = '0' else                    } 器件 mux21 的内部工作逻辑描述，即实体描述的器件功能结构，这一部分称为结构体
         b when s = '1';                        }
end architecture one;                           }
```

这是一个完整的2选1多路选择器的VHDL文件。VHDL编译器和综合器可以独立地对它进行编译和综合，对于综合后得到的标准格式网表文件，如edif文件，可用于通过针对特定的目标芯片，如Altera公司的某一器件EPM7128S进行适配，由此可获得对应的仿真文件和编程下载文件。前者可用于对程序3-1的设计进行仿真测试，以便了解其逻辑功能是否满足原设计的要求，而后者是对器件EPM7128S的编程文件可用于实现硬件功能和完成硬件测试(hardware debug)，即可利用某个EDA平台例如Quartus Ⅱ(详细使用方法可参考第4章)，将此VHDL文件进行编译综合等处理。然后将mux21的4个引脚信号a、b、s和y锁定于某个具体的目标芯片引脚上，再进行映射适配，即利用计算机在Quartus Ⅱ的帮助下将程序文件综合后得到的网表文件，配置进该选定的FPGA或CPLD器件中，最后将所得的配置文件编程下载进这一芯片中，这时芯片就有了如程序所描述的2选1选择器mux21的功能。如果对这片赋予了2选1逻辑功能的器件进行实际的测试，即为硬件仿真。集成电路厂商能够很容易地将符合工业标准的VHDL所描述的逻辑设计文件综合成可映射于半导体门阵列的网表文件。但是，如果仅仅准备将此芯片直接用于产品的电路板上，这个测试过程只能称为硬件调试。

从上例文件的描述层次上来看，选择器整体设计的VHDL描述使用了以下3个层次。

1. 库(library)说明

库包含了描述器件的输入/输出端口数据类型(即端口信号的取值类型或范围)中将要用到的IEEE的标准库中的std_logic_1164程序包。

明确地指定和严格地定义端口信号的取值类型是VHDL的重要特点，此所谓强类型语言，这是学习VHDL特别应当注意的地方。

2. 实体(entity)说明

实体的电路意义相当于器件，在电路原理图上相当于元件符号。实体是一个完整的、独立的语言模块，它描述了mux21接口信息，定义了器件mux21端口引脚a、b、s和y的输入/输出性质和数据类型，它利用port语句说明了mux21的外部引脚的工作方式，所以port的描述相当于电路器件的外部引脚。in对端口引脚a和b进行了信号流向的方向说明，它规定了信号必须由外部通过端口引脚a、b流进所描述的器件内部；而out则规定了器件内部的信号需通过端口引脚y向外输出。同时指明了端口a、b、s和y信号的数据类型符合ieee库中std_logic_1164程序包中的标准数据位std_logic中所定义的数据类型。

3. 结构体(architecture)说明

结构体层次描述了mux21内部的逻辑功能，在电路上相当于器件的内部电路结构。此例的逻辑描述十分简洁，它并没有将选择器内部逻辑门的连接方式表达出来，而是将此选择器看成一个黑盒，以类似于计算机高级语言的表达方式描述了它的外部逻辑行为。符号“<=”是信号赋值符，是信号传递的意思，“y<=a”表示将a获得的信号赋给y输出端，这是一个单向过程。

需要指出的是，程序中实体和结构体分别是以“end (entity) mux21”和“end (architecture) one”语句结尾的，这是符合VHDL新版本IEEE STD 1076—1993的语法

要求的。若根据 VHDL'87 版本，IEEE STD 1076—1987 的语法要求这两条结尾语句只需写成“end mux21”和“end one”。但考虑到目前绝大多数常用的 EDA 工具中的 VHDL 综合器仍以支持 VHDL'87 版本所有语法规则为主，且许多最新的 VHDL 方面的资料仍然使用 VHDL'87 版本语言规则，因此，出于实用的目的，对于以后出现的示例，不再特意指出 VHDL 两种版本的语法差异处。

一个可综合的 VHDL 描述的最小和最基本的逻辑结构中，IEEE 标准库说明、实体和结构体是最基本的和不可缺少的 3 个部分，其他的结构层次可根据需要选用。2 选 1 程序作为一个完整的 VHDL 描述既可以作为一个独立的功能器件使用和保存，也能被其他的由 VHDL 描述的逻辑电路所调用，成为其中的一个功能部件。

从程序可以清晰地看出，一个完整的 VHDL 描述是以对一个功能元件的完整描述为基础的，因此元件是 VHDL 的特定概念，也是 VHDL 的鲜明特色，把握了元件的结构和功能的完整描述，就把握了 VHDL 的基本结构。元件本身具有层次性，即任一元件既可以是单一功能的简单元件，也可以是由许多元件组合而成的，具有更复杂功能的元件，乃至一个电路系统。但从基本结构上看，都能用上述程序给出的 3 个部分来描述，这种元件概念的直观性是其他 HDL 所无法比拟的，它为自顶向下或自下向上灵活的设计流程奠定了坚实的基础。

一个完整的 VHDL 语言程序通常包含实体(entity)、结构体(architecture)、配置(configuration)、程序包(package)和库(library)5 个部分。前 4 个部分是可分别编译的源设计单元。在 VHDL 程序中，实体和结构体这两个基本结构是必需的，它们可以构成最简单的 VHDL 程序，如上述的 2 选 1 选择器 VHDL 程序。实体用于描述所设计的系统的外接口信号；结构体用于描述系统内部的结构和行为；程序包存放各种设计模块都能共享的数据类型、常数和子程序等；配置用于从库中选取所需单元来组成系统设计的不同版本；库存放已经编译的实体、构造体、程序包和配置。库可由用户生成或由 ASIC 芯片制造商提供，以便于在设计中为大家所共享。

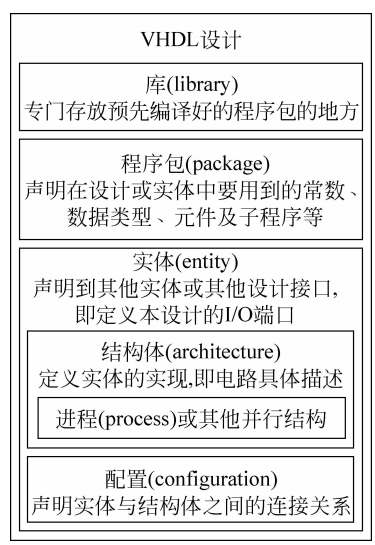

图 3-2　VHDL 程序结构示意图

VHDL 程序结构可以用图 3-2 表示，一个完整的 VHDL 设计必须包含一个实体和一个与之相对应的结构体。一个实体可以对应多个结构体，以表示可采用不同的方法来描述电路，但要通过配置语句进行说明。如果编程时使用了系统资源，则需要引用库和调用程序包。

3.2.2　实体

实体是 VHDL 程序设计的最基本模块，简单的可以是一个与门，复杂的可以是一个数字系统或微处理器，其结构基本一致。

1. 实体语句结构

以下是实体说明单元的常用语句结构：

```
entity 实体名 is
  [generic(类属表);]
  port ( 端口表 );
end entity 实体名;
```

实体说明单元必须按照这一结构来编写，实体应以语句“entity 实体名 is”开始，以语句“end entity 实体名；”结束。其中的实体名可以由设计者自己添加，中间在方括号内的语句描述在特定的情况下并非是必需的。例如构建一个 VHDL 仿真测试基准等情况中，可以省去方括号中的语句。对于 VHDL 的编译器和综合器来说，程序文字的大小写是不加以区分的。有的教材为了便于阅读和分辨，将 VHDL 的标识符或基本语句关键词以大写方式表示，而由设计者添加的内容可以以小写方式来表示，如实体的结尾写为“end entity nand”，其中的 nand 即为设计者取的实体名。当然，也可以根据自己的喜好将 VHDL 程序中的英文字母全部大写或全部小写都可以，不影响利用 EDA 工具进行编译综合。本书的 VHDL 程序照顾大多数人的阅读习惯，英文字母全部小写。

2. 实体名

实体名具体是由设计者自己确定的。由于实体名实际表达的是该 VHDL 设计的电路器件名称，因此最好根据相应的电路功能来确定。如 4 位二进制计数器，实体名可取为“counter4b”；8 位二进制加法器，实体名可取为“adder8b”。

另外，在给实体名命名时（包括其他一些命名），有一些基本规则要遵循。不能用纯数字或中文定义实体名，也不能用 EDA 工具库中已定义好的元件名、运算操作符等关键字（或叫保留字）作为实体名称，如 or2、latch 等，且不能用数字起头命名实体名，如 74LS××等。

3. 类属说明语句

类属（generic）参量是一种端口界面常数，常以一种说明的形式放在实体或块结构体前的说明部分，类属为所说明的环境提供了一种静态信息通道。类属与常数不同，常数只能从设计实体的内部得到赋值，且不能再改变，而类属的值可以由设计实体外部提供。因此，设计者可以从外面通过类属参量的重新设定而方便地改变一个设计实体或一个元件的内部电路结构和规模。

类属说明的一般书写格式如下：

```
generic(常数名 1: 数据类型 1: 设定值 1 ;
                    ...
        常数名 n: 数据类型 n: 设定值 n);
```

类属参量以关键词 generic 引导一个类属参量表，在表中提供时间参数或总线宽度等静态信息。类属表说明用于设计实体和其外部环境通信的参数，传递静态的信息。类属在所定义环境中的地位与常数十分接近，但却能从环境如设计实体外部动态地接受赋值，其行为又有点儿类似于端口 port。因此常如以上的实体定义语句那样，将类属说明放在其中，且放在端口说明语句的前面。

在一个实体中定义的、来自外部赋入类属的值，可以在实体内部或与之相对应的结构体中读到。对于同一个设计实体，可以通过 generic 参数类属的说明，为它创建多个行为不同的逻辑结构。比较常见的情况是利用类属来动态规定一个实体的端口大小，或设计实体的物理特性，或结构体中的总线宽度，或设计实体底层中同种元件的例化数量等。

一般在实体中，类属的应用与常数是一样的。例如当用实体例化一个设计实体的器件时，可以用类属表中的参数项定制这个器件，如可以将一个实体的传输延迟、上升和下降延迟等参数加到类属参数表中，然后根据这些参数进行定制，这对于系统仿真控制是十分方便的。其中的常数名是由设计者确定的类属常数名，数据类型通常取 integer 或 time 等类型，设定值即为常数名所代表的数值。但需注意 VHDL 综合器仅支持数据类型为整数的类属值。

有类属说明的 2 输入与非门的实体描述如下。

【程序 3-2】 有类属说明的 2 输入与非门的实体。

```
entity nand2 is
   generic(t_rise : time: =2ns;
           t_fall : time: =1ns);
   port(a, b : in bit;
           s : out bit);
end nand2;
```

在这个例子中，generic 类属语句对一个 2 输入与非门的实体上升沿和下降沿时间做了具体定义，对于类属值 t_rise、t_fall 的改变将决定这个设计实体进行仿真时的结果。对于本例来说，改变类属值不会改变整个设计实体的硬件结构，因为 VHDL 综合器仅仅支持数据类型为整数的类属值，也就是说数据类型为 time 的类属语句不能被综合而得到与之相对应的硬件结构，在综合时会被忽略。但是，数据类型为整数的类属语句是可以被综合的，因此对于这样的设计实体，如果改变类属值就会改变整个设计实体的硬件结构。

4. port 端口说明

由 port 引导的端口说明语句是对一个设计实体界面的说明，其端口表部分对设计实体与外部电路的接口通道进行了说明，其中包括对每一接口的输入/输出模式（mode 或称端口模式）和数据类型 type 进行了定义。在实体说明的前面，可以有库的说明，即由关键字 library 和 use 引导一些对库和程序包使用的说明语句，其中的一些内容可以为实体端口数据类型的定义所用。

实体端口说明的一般书写格式如下：

```
PORT（端口名 : 端口模式 数据类型;
      端口名 : 端口模式 数据类型);
```

其中的端口名是设计者为实体的每一个对外通道所取的名字，端口模式是指这些通道上的数据流动方式，如输入或输出等。数据类型是指端口上流动数据的表达格式或取值类型，这是由于 VHDL 是一种强类型语言，即对语句中所有的端口信号、内部信号和操作数的数据类型有严格的规定，只有相同数据类型的端口信号和操作数才能相互作用。

一个实体通常有一个或多个端口，端口类似于原理图中元件符号上的引脚。实体与

外界交流的信息必须通过端口通道流入或流出。程序 3-3 是一个 2 输入与非门的实体描述示例，图 3-3 是它对应的原理图符号。

【程序 3-3】 2 输入与非门实体描述。

```
library ieee;
use ieee.std_logic_1164.all;
entity nand2 is
  port(a : in std_logic;
       b : in std_logic;
       c : out std_logic);
end nand2;
...
```

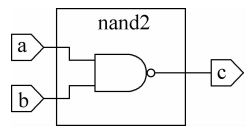

图 3-3 nand2 对应的原理图符号

图 3-3 中的 nand2 可以看成一个设计实体，它的外部接口界面由输入/输出信号端口 a、b 和 c 构成，内部逻辑功能是一个与非门。在电路图上，端口对应于器件符号的外部引脚。端口名作为外部引脚的名称，端口模式用来定义外部引脚的信号流向。IEEE 1076 标准程序包中定义了常用的端口模式。

在实际的数字集成电路中，in 相当于只可输入的引脚，out 相当于只可输出的引脚，buffer 相当于带输出缓冲器并可以回读的引脚（与 tri 引脚不同），而 inout 相当于双向引脚（即 bidir 引脚），是普通输出端口（out）加入三态输出缓冲器和输入缓冲器构成的。表 3-1 列出了端口的功能。

表 3-1 端口模式说明

端 口 模 式	端口模式说明（以设计实体为主体）
in	输入，只读模式
out	输出，单向赋值模式
buffer	具有读功能的输出模式，从内部看可以读或写，只能有一个驱动源
inout	双向，从内部或外部看都可以读或写

3.2.3 结构体

由图 3-2 可知，结构体是实体所定义的设计实体中的一个组成部分。结构体描述设计实体的内部结构以及与外部设计实体端口间的逻辑关系，结构体由如下两大部分组成。

（1）对数据类型、常数信号、子程序和元件等元素的说明部分。

（2）描述实体逻辑行为，以各种不同的描述风格表达功能描述语句，它们包括各种形式的顺序描述语句和并行描述语句。

结构体将具体实现一个实体。每个实体可以有多个结构体，每个结构体对应着实体不同的结构和算法实现方案，其间各个结构体的地位是同等的，它们完整地实现了实体的行为，同一结构体不能为不同的实体所拥有。结构体不能单独存在，必须有一个界面说明，即一个实体。对于具有多个结构体的实体，必须用 configuration 配置语句指明用于综合的结构体和用于仿真的结构体。即在综合后的可映射于硬件电路的设计实体中，一个实体只能对应一个结构体。在电路中，如果实体代表一个器件符号，则结构体描述了这个符号的内部行为。当把这个符号例化成一个实际的器件安装到电路上时，则需配置语

句为这个例化的器件指定一个结构体(即指定一种实现方案),或由编译器自动选一个结构体。

1. 结构体的一般语句格式

结构体的语句格式如下:

```
architecture 结构体名 of 实体名 is
  [说明语句]
begin
  [功能描述语句]
end (architecture) 结构体名;
```

在书写格式上,实体名必须是所在设计实体的名字,而结构体名可以由设计者自己选择,但当一个实体具有多个结构体时,结构体的取名不可相重。结构体的说明语句部分必须放在关键词"architecture"和"begin"之间,结构体必须以"end (architecture) 结构体名;"作为结束句。

结构体内部构造的描述层次和描述内容可以用图 3-4 来说明。此图只是对结构体的内部构造进行了一般的描述,并非所有的结构体必须同时具有如图 3-4 所示的所有说明语句结构。一般而言,一个完整的结构体由两个基本层次组成,即说明语句和功能描述语句两部分。

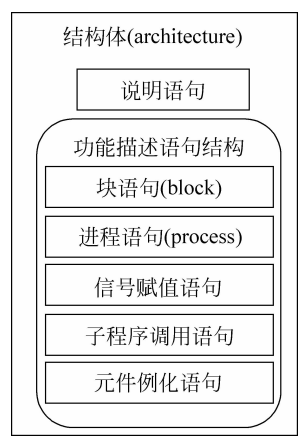

图 3-4　结构体构造图

2. 结构体说明语句

结构体中的说明语句是对结构体的功能描述语句中将要用到的信号(signal)、数据类型(type)、常数(constant)、元件(component)、函数(function)和过程(procedure)等加以说明。注意:在一个结构体中,说明和定义的数据类型、常数、元件、函数和过程只能用于这个结构体中。如果希望这些定义也能用于其他的实体或结构体中,需要将其作为程序包来处理。

3. 功能描述语句结构

图 3-4 所示的功能描述语句结构可以含有 5 种不同类型的以并行方式工作的语句结构,这可以看成是结构体的 5 个子结构。而在每一语句结构的内部,可能含有并行运行的逻辑描述语句或顺序运行的逻辑描述语句。也就是说,这 5 种语句结构本身是并行语句,但它们内部所包含的语句并不一定是并行语句,如进程语句内所包含的是顺序语句。

图 3-4 中的 5 种语句结构的基本组成和功能如下。

(1) 块语句是由一系列并行执行语句构成的组合体,它的功能是将结构体中的并行语句组成一个或多个子模块。

(2) 进程语句定义顺序语句模块,用于将从外部获得的信号值,或内部的运算数据向其他的信号进行赋值。

(3) 信号赋值语句将设计实体内的处理结果向定义的信号或界面端口进行赋值。

(4) 子程序调用语句用于调用过程或函数,并将获得的结果赋值于信号。

(5) 元件例化语句对其他的设计实体进行元件调用说明,并将此元件的端口与其他的元件、信号或高层次实体的界面端口进行连接。

下面的程序 3-4 是一个 2 输入与门结构体的描述,它的结构体名是 behav,结构体内有一个进程语句子结构。在此结构中用顺序语句描述了与门的输入信号 a0 和 a1 与输出信号 z0 之间的逻辑关系,以及它们的时延关系。

【程序 3-4】 2 输入与门结构体描述语句参考程序。

```
library ieee;
use ieee.std_logic_1164.all;
entity pgand2 is
    generic (trise : time := 1 ns;
             tfall : time := 1 ns);
   port (a1:in std_logic;
         a0:in std_logic;
         z0:out std_logic);
end entity pgand2;
architecture behave of pgand2 is
begin
   process (a1, a0)
     variable zdf:std_logic;
    begin
      zdf := a1 and a0;                    --向变量赋值
      if zdf ='1' then
        z0 <= zdf after trise;
      elsif zdf ='0' then
        z0 <= zdf after tfall;
      else
        z0 <= zdf;
      end if;
   end process;
end architecture behav;
```

注意:VHDL 综合器将不支持或忽略此例中的时延关系,如"after tfall"。两条短画线是注释说明,其右侧的内容是对程序的具体解释,并不执行。

3.2.4 库、程序包和配置

1. 库(library)

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好的数据类型、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、数据集合体和元件的仓库。如果要在一项 VHDL 设计中用到某一程序包,就必须在这项设计中预先打开这个程序包,使此设计能随时使用这一程序包中的内容。在综合过程中,每当综合器在较高层次的 VHDL 源文件中遇到库语言,就将随库指定的源文件读入,并参与综合。也就是说,在综合过程中,所要调用的库必须以 VHDL 源文件的方