# 第3章 chapter 3

# VHDL语言基础

本章详细介绍VHDL语言的基本结构、VHDL语言要素、VHDL语言语句的原理和设计方法。VHDL语言是整个EDA设计中最核心的内容之一。读者必须熟练掌握VHDL语言，并且通过实验掌握使用VHDL语言对可编程逻辑器件进行编程的方法和技巧。

## 3.1 VHDL程序结构

### 3.1.1 VHDL程序结构概述

一个完整的VHDL程序包含实体(entity)、结构体(architecture)、配置(configuration)、包集合(package)、库(library)5个部分。实体主要是用于描述和外部设备的接口信号；构造体用于描述系统的具体逻辑行为功能；包存放设计使用到的公共的数据类型、常数和子程序等；配置用来从库中选择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、包集合和配置等。

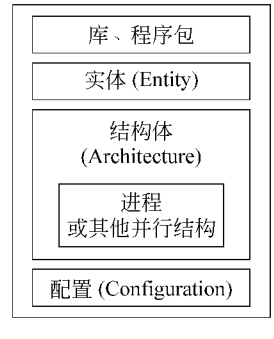



**图 3.1 VHDL程序的结构图**

VHDL的基本结构由实体和结构体两部分组成。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。根据所设计数字系统的复杂度不同，其程序规模也大不相同。图3.1给出了VHDL程序的基本结构示意图，该图表明了一个完整的VHDL程序所应该包含的部分。

### 3.1.2 VHDL程序实体

实体由类属说明和端口说明两个部分组成。根据IEEE标准，VHDL程序实体的一般格式为：

```
entity <entity_name> is
```

```
generic (
 <generic_name>: <type>:=<value>;
 <other generics>...
);
port (
 <port_name>: <mode><type>;
  <other ports>...
 );
end <entity_name>;
```

实体说明在VHDL程序设计中描述一个元件或一个模块与设计系统的其余部分(其余元件、模块)之间的连接关系,可以看作一个电路图的符号。因为在一张电路图中,某个元件在图中与其他元件的连接关系是明显直观的。

**1. 类属说明**

类属说明是实体说明中的可选项,放在端口说明之前,用于指定参数,其一般书写格式为:

```
generic (
 <generic_name>: <type>:=<value>;
 <other generics>...
);
```

类属说明常用来定义实体端口的大小、设计实体的物理特性、总线宽度、元件例化的数量等。

**2. 端口说明**

定义实体的一组端口称作端口说明(port declaration)。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,也可以当作变量用在逻辑表达式中。

端口说明结构必须有端口名、端口方向和数据类型。端口说明的一般格式为:

```
port (
  <port_name>: <mode><type>;
  <other ports>...
);
```

1) 端口名

端口名(port_name)是赋予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。

2）模式

模式(mode)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) 输入模式。

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(in)主要用于时钟输入、控制输入(如 Load、Reset、Enable、CLK)和单向的数据输入，如地址信号(address)。

(2) 输出模式。

输出仅允许数据流从实体内部输出。如图 3.2(a)所示，端口的驱动源是由被设计的实体内部进行的。输出模式(out)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作可读的。输出模式常用于计数输出、单向数据输出及设计实体产生的控制其他实体的信号等。

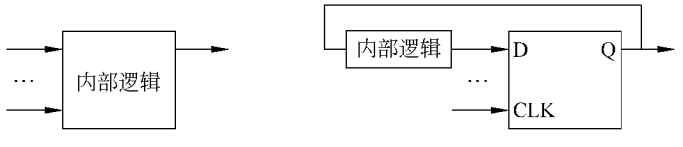
(3) 缓冲模式。

缓冲模式(buffer)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或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如下：

- 建立缓冲模式端口。
- 建立设计实体的内部节点。

如图 3.2(b)所示，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。




(a) 输出模式　　(b) 缓冲模式

**图 3.2　输出模式和缓冲模式的区别**

(4) 双向模式。

双向模式(inout)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。典型的，常见的 SRAM 和 SDRAM 芯片的数据端口就是双向的。在第 4 章将详细地说明对双向端口的控制方法。

3）数据类型

数据类型(type)端口说明除了定义端口标识名称、端口定义外，还要标明出入端口的

数据类型。VHDL语言的标准规定，EDA综合工具支持的数据类型为布尔型(boolean)、位型(bit)、位矢量型(bit-vector)和整数型(integer)。

为了使EDA工具的仿真、综合软件能够处理这些逻辑类型，这些标准库必须在实体中声明或在USE语句中调用。

【例3-1】 下面给出一个关于8位计数器的实体说明

```
entity counter is
generic (byte:  integer:=8);
port(
      clk      :    in  std_logic;
      rst      :    in  std_logic;
      counter  :    out std_logic_vector(byte-1 downto 0)
    );
end counter;
```

在上面的例子中描述了一个8位的计数器的实体部分。该实体部分包含类属说明和端口说明。从例子中可以看到，在实体中声明了一个byte的类属名，该类属表示8个比特位。因此，在counter端口的类型说明中直接使用在该实体中所定义的byte类属名。在实体的端口说明部分，说明了3个端口clk、rst、counter。其中clk、rst均为输入端口，而counter为输出端口。需要注意的是，在实体中使用类属在很多情况下是为了修改程序的方便。

### 3.1.3 VHDL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逻辑功能和行为，规定了该设计实体的内部模块及其内部模块的连接关系。VHDL对构造体的描述通常有三种方式进行描述：行为描述、寄存器传输描述和结构描述，这三种描述方式将在后面进行详细的说明。

由于结构体是对实体功能的具体描述，所以结构体一定在实体的后面。

一个结构体的VHDL的描述为：

```
architecture <arch_name>of <entity_name>is
   --declarative_items (signal declarations, component declarations, etc.)
begin
   --architecture body
end <arch_name>;
```

其中，arch_name为结构体的名字；entity_name为实体的名字。

结构体的begin开始的前面部分为声明项(declarative_items)，通常是对设计内部的信号或者元件进行声明；而begin后面一直到结构体的结束，该部分是对实体行为和功能的具体的描述。该部分的描述是由顺序语句和并发语句完成的。

#### 1. 结构体命名

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 rtl OF mux IS              —用结构体的寄存器传输结构命名
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 behave OF mux IS           —用结构体的行为描述方式命名
```

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。命名时应该简明扼要，一目了然。用 VHDL 写的文档不仅是 EDA 工具编译的源程序，而且对于一个完整的设计来说也是非常重要的。对于一个从事 EDA 设计的读者来说，培养一个良好的命名习惯可以在某种程度上提高设计效率，同时也有利于整个项目的管理。

### 2. 结构体内信号定义

由结构体的书写格式知道，在关键字 ARCHITECTURE 和 BEGIN 之间的部分，用于对结构体内部使用的信号、常数、数据类型和函数进行定义。

特别需要注意的是，这些声明用于结构体内部，而不能用于实体内部，因为一个实体可能有几个结构体相对应。另外，实体说明中定义 I/O 信号为外部信号，而结构体定义的信号为内部信号。

结构体的信号定义和实体的端口说明一样，应有信号名称和数据类型定义，但不需要定义信号模式(mode)，不需要说明信号方向，因为这些结构体的内部信号是用来描述结构体内部的连接关系。

**【例 3-2】** 结构体的内部信号定义方法

```
ARCHITECTURE structural OF mux IS
  signal   a,b:  std_logic;
  signal    x:  std_logic_vector(0 to 7);
  signal    y:  integer 0 to 255;
BEGIN
  ⋮
END structural;
```

从上面的例子可以看出，在结构体内声明的信号不需要指明端口的方向，因为在结构体内部的信号只是用来表示设计实体内部的逻辑之间的连接关系。

### 3. 结构体并行处理语句

并行处理语句是结构体描述的主要语句，并行处理语句在 begin 和 end 之间。并行处理语句表明，若一个结构体的描述用的是结构描述方式，则并行语句表达了结构体的内部元件之间的互连关系。这些语句是并行的，各个语句之间没有顺序关系。

若一个结构体是用进程语句来描述的，并且这个结构体含有多个进程，则各进程之间是并行的。但必须声明，每个进程内部的语句是有顺序的，不是并行的。

若一个结构体用模块化结构描述，则各模块间是并行的，而模块内部视描述方式而定。

【例 3-3】 用并行语句描述的结构体

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux4 IS
  PORT (a    :   in  std_logic_vector(3 downto 0);
        sel  :   in  std_logic_vector(1 downto 0);
        q    :   out  std_logic);
END mux4;
ARCHITECTURE rtl OF mux IS
BEGIN
    q<=a(0)  when sel="00" else
       a(1)  when sel="01" else
       a(2)  when sel="10" else
       a(3)  when sel="11" else
       'X'
END rtl;
```

在上面的例子中，使用了条件带入语句来描述了一个四选一的逻辑单元，该条件信号带入语句是并发描述语句。下面将对该条件带入语句进行详细的描述。

## 3.2 VHDL语言描述风格

VHDL语言主要有3种描述风格：行为描述、数据流(RTL寄存器传输)描述和结构描述。这3种描述方式从不同角度对硬件系统进行描述。一般情况下，行为描述用于模型仿真和功能仿真；而RTL描述和结构描述可以进行逻辑综合。

### 3.2.1 结构体行为描述

行为描述是以算法形式对系统模型、功能的描述，与硬件结构无关。抽象程度最高。行为描述中常用的语句主要有进程、过程和函数。

【例 3-4】 两输入或门的行为描述

```
ENTITY and2 IS
  PORT (a, b: in std_logic;
        c: out std_logic);
END and2;
ARCHITECTURE behav of and2 is
BEGIN
   c<=a or b AFTER 5 ns;
END behave;
```

### 3.2.2 结构体数据流描述

数据流描述又称为寄存器传输级 RTL 描述。RTL 级描述是以寄存器为特征，在寄存器之间插入组合逻辑电路，即以描述数据流的流向为特征。图 3.3 给出了结构体的数据流的图形描述。

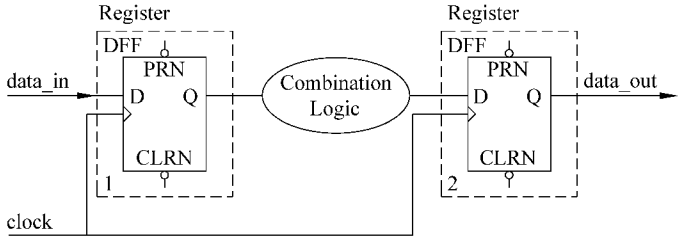



**图 3.3 结构体的数据流描述**

**【例 3-5】** 四选一选择器的数据流(RTL)描述

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY mux4 IS
  port (x   :    in   std_logic_vector(3 downto 0);
        sel :    in   std_logic_vector(1 downto 0);
        y   :    out  std_logic);
END mux4;
ARCHITECTURE rtl of mux4 IS
BEGIN
  y<=x(0) when sel="00" else
     x(1) when sel="01" else
     x(2) when sel="10" else
     x(3);
END rtl;
```

这种基于 RTL 级的描述，虽然具体了一些，但仍没有反映出实体内的具体结构。使用 RTL 描述时，应遵循以下几个原则：

(1) 在一个进程中，不允许存在两个寄存器的描述。

**【例 3-6】** 违反规则(1)的描述

```
PROCESS(clk1,clk2)
BEGIN
 if rising_edge(clk1) then
    y<=a;
 end if;
   if rising_edge(clk2) then
```

```
    z<=b;
  end if;
END PROCESS;
```

上面的例子中，在一个进程中使用了两个时钟，激励两个触发器，这是违反规则(1)的。

(2) 在描述寄存器时，不允许使用 IF 语句中的 ELSE 项。

**【例 3-7】** 违反规则(2)的描述

```
PROCESS(clk)
BEGIN
 if rising_edge(clk) then
    y<=a;
 else
    y<=b;
 end if;
END PROCESS;
```

(3) 在描述寄存器时，必须带入信号值。

**【例 3-8】** 带入信号的方法

```
PROCESS(clk)
 VARIABLE tmp:  std_logic;
BEGIN
   y<=tmp;
if rising_edge(clk) then
    tmp:=a;
 end if;
END PROCESS;
```

### 3.2.3 结构体结构化描述

在多层次设计中，高层次的设计模块调用低层次的设计模块，构成模块化的设计。从下面的例子可以看出来，全加器由两个半加器和一个或门构成，元件之间、元件与实体端口之间通过信号连接。知道了它们的构成方式，那么就可以通过元件例化语句进行描述。

**【例 3-9】** 全加器的结构化的 VHDL 的描述

图 3.4 给出了全加器的结构化的图形描述。

```
Architecture structure_view of Full_adder is
   Component half_adder
   port(a,b: in std_logic; s,c: out std_logic);
end component;
component or_gate
   port(a,b: in std_logic; c: out std_logic);
```

```
end component;
  signal a,b,c: std_logic;
begin
Inst_half_adder1:    port map(x,y,a,b);
Inst_half_adder2:    port map(a,cin,sum,c);
Inst_or_gate:        port map(b,c,cout);
End structure_view;
```

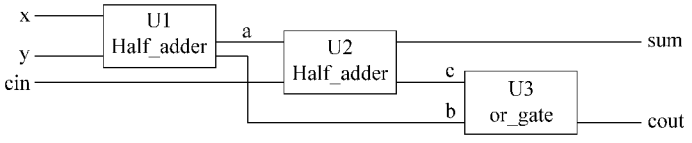



**图 3.4　全加器的结构化描述**

结构层次化编码是模块化设计思想的一种体现。目前大型设计中必须采用结构层次化编码风格，以提高代码的可读性，易于分工协作，易于设计仿真测试激励。最基本的结构化层次是由一个顶层模块和若干个子模块构成，每个子模块根据需要还可以包含自己的子模块。结构层次化编码结构如图 3.5 所示。

在进行结构层次化设计过程中，要遵循以下的原则：

(1) 结构的层次不易太深，一般为 3～5 层即可。在综合时，一般综合工具为了获得更好的综合效果，特别是为了使综合结果所占用的面积更小，会默认将 RTL 代码的层次打平。而有时为了在综合后仿真和布局布线后时序仿真中较方便的找出一些中间信号，比如子模块之间的接口信号等，可以在综合工具中设置保留结构层次，以便于仿真信号的查找和观察。

(2) 顶层模块最好仅仅包含对所有模块的组织和调用，而不应该完成比较复杂的逻辑功能。较为合理的顶层模块由输入输出管脚声明、模块的调用与实例化、全局时钟资源、全局置位/复位、三态缓冲和一些简单的组合逻辑等构成。

(3) 所有的 I/O 信号，如输入、输出、双向信号等的描述在顶层模块中完成。

(4) 子模块之间也可以有接口，但是最好不要建立子模块间跨层次的接口，例如图 3.5 中模块 A1 到模块 B1 之间不宜直接连接，两者需要交换的信号可以通过模块 A、模块 B 的接口传递。这样做的好处是增加了设计的可读性和可维护性。

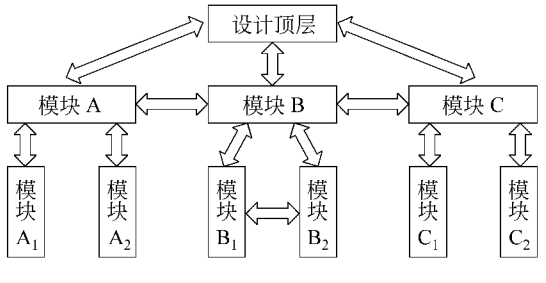



**图 3.5　结构层次化设计示意图**

(5) 子模块的合理划分非常重要,应该综合考虑子模块的功能、结构、时序、复杂度等多方面因素。

## 3.3 设计资源共享

除实体和构造体外,包集合、库及配置是 VHDL 语言另外 3 个可以各自独立进行编译的源设计单元。通过使用库、包和配置,可以实现设计的共享。

### 3.3.1 库

**1. 库的种类**

一个库中可以存放集合定义、实体定义、结构体定义和配置定义。当需要引用一个库时,首先需要对库名进行说明,其格式为:

```
LIBRARY  <library_name>
```

其中,<library_name>为库的名字,这时就可以使用库中已经编译好的设计。对库中集合包的访问必须再经由 USE 语句才能打开。其格式为:

```
USE  <package_name>
```

其中,<package_name>为程序包的名字。

当前在 VHDL 语言中的库大致可以分为 5 种:IEEE 库、STD 库、ASIC 矢量库、用户定义库和 WORK 库。

1) IEEE 库

定义了 4 个常用的程序包:

- std_logic_1164 (std_logic types & related functions)
- std_logic_arith (arithmetic functions)
- std_logic_signed (signed arithmetic functions)
- std_logic_unsigned (unsigned arithmetic functions)

2) STD 库(默认库)

STD 库是 VHDL 的标准库,在库中存放 STANDARD 的包集合。由于它是 VHDL 的标准库,因此设计人员如果调用 STANDARD 包内的数据可以不进行标准格式的说明。STD 库中还包含 TEXTIO 的包集合,在使用这部分包时,必须说明库和包集合名,然后才能使用该包集合中的数据。

```
LIBRARY STD
USE STD.TEXTIO.ALL;
```

STANDARD 包集合中定义了最基本的数据类型,包括 Bit、bit_vector、Boolean、Integer、Real 和 Time 等。