VHDL语言及其应用

缪善林

QQ：330495908

群：32998872

哈尔滨工程大学信息与通信工程学院

第一部分 VHDL综述(1)

- 什么是VHDL语言？
  - HDL语言是一种支持用形式化方法来描述数字逻辑电路和系统的语言
  - VHDL语言源于美国国防部发起的VHSIC(Very High Speed Integrated Circuits)计划
  - 1987年12月IEEE批准VHDL为标准HDL语言(IEEE-1076)，称为VHDL’87
  - 1993年修订为VHDL’93， 2001年修订为VHDL 2001
  - IEEE-1076.1 VHDL-AMS, IEEE-1076.2 Mathematical Packages, IEEE-1076.3 Synthesis Packages, IEEE-1076.4 VITAL, IEEE Standard 1164 Multi-value Logic System
- VHDL语言的优点？
  - 人机可读性好
  - 比图形和布尔方程更简洁
  - 方便设计重用
  - 容易实现设计仿真与验证
  - 便于映射为IC芯片的制造工艺

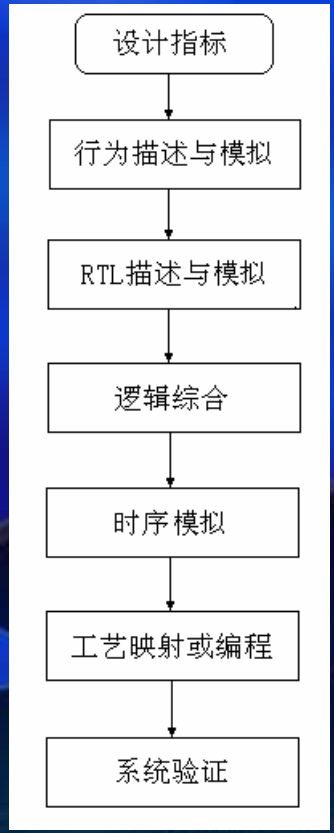
第一部分 VHDL综述(2)

- 自顶向下设计
  - VHDL语言支持自顶向下的系统划分，直至划分后的最底层单元能用图元(primitive element)来实现为止
  - 图元就是基本逻辑单元、宏模型或IP_core

- VHDL的基本模型
  - 行为模型：用于描述数字器件或系统的功能，统指数学方程表示的模型
  - 时序模型：用于描述数字器件或系统的激励与响应间的关系，统指布尔方程表示的模型
  - 结构模型：用于描述自顶向下划分系统形成的各个基本单元间的互连关系，统指用元件互连生成的电路模型

第一部分 VHDL综述(3)

- VHDL的基本设计流程



第一部分 VHDL综述(4)

- **VHDL的主要应用领域**
  - ▲智能模块(IP)的研发
    - **IP：用VHDL语言编写，经逻辑优化和功能验证的可生成VLSI中各种功能单元的软件群，例如，无线通信产品、网上设备、中央处理器(通用CPU)、DSP、PCI、USB、嵌入式CPU**
  - ▲单芯片全功能集成系统设计：**SoPC系统、嵌入式计算、ASIC验证**
  - ▲功能可重置系统的设计：远程系统升级、可重配置设计
- **FPGA的发展方向**
  - ▲多用途：**CAM，RAM、PLL**
  - ▲高密度、低功耗、低成本：***MAX* II系列、Cyclone II系列**
  - ▲嵌入硬核、超高速：**Stratix系列、Stratix II系列、Stratix GX系列**
  - ▲超低成本：结构化的**ASICs，例如，HardCopy系列**
  - ▲嵌入CPU软核：**Nios、Nios II**

第一部分 VHDL综述(5)

- Altera公司的最新FPGA芯片

Table 17. Stratix II Devices

DEVICE	ADAPTIVE LOGIC MODULES (ALMS)[1]	EQUIVALENT LEs[1]	PIN/PACKAGE OPTIONS	MAXIMUM USER I/O PINS	SUPPLY VOLTAGE	M512 RAM BLOCKS	M4K RAM BLOCKS	M-RAM BLOCKS	TOTAL RAM BITS	DSP BLOCKS	EMBEDDED MULTIPLIERS[2]	PLLs[3]
EP2S15	6,240	15,600	484-Pin BGA[4], 672-Pin BGA[4]	341 365	1.2 V	104	78	0	419,328	12	48	6
EP2S30	13,552	33,880	484-Pin BGA[4], 672-Pin BGA[4]	341 499	1.2 V	202	144	1	1,369,728	16	64	6
EP2S60	24,176	60,440	484-Pin BGA[4], 672-Pin BGA[4], 1,020-Pin BGA[4]	341 499 717	1.2 V	329	255	2	2,544,192	36	144	12
EP2S90	36,384	90,960	1,020-Pin BGA[4], 1,508-Pin BGA[4]	757 901	1.2 V	488	408	4	4,520,448	48	192	12
EP2S130	53,016	132,540	1,020-Pin BGA[4] 1,508-Pin BGA[4]	741 1,109	1.2 V	699	609	6	6,747,840	63	252	12
EP2S180	71,760	179,400	1,020-Pin BGA[4] 1,508-Pin BGA[4]	741 1,173	1.2 V	930	768	9	9,383,040	96	484	12

Notes to Table 17:

[1]Each Stratix II ALM is equivalent to 2.5, 4-input look-up table (LUT)-based LEs.
[2]Each DSP block supports four 18×18 multipliers.
[3]Includes enhanced and fast PLLs.
[4]Space-saving FineLine BGA package

第二部分 VHDL语言的学习基础(1)

- 书写规定与基本句法单元
  - ▲ 书写规定

保留字	用黑体小写字母表示
类型字	用小写字母表示
库	用大写字母表示
标识符	用小写字母表示
简化书写	用〈...〉表示
任选项	用［...］表示
重复项	用｛...｝表示，有时也用之界定一段语句
二选一	用 ...|... 表示
定义为	用 ::= 表示
语句分隔	用分号“；”表示
特殊要强调的内容	用黑体表示
注释	用“--”前缀

第二部分 VHDL语言的学习基础(2)

- 书写规定与基本句法单元

▲ 标识符

基本标识符 由VHDL'87支持，长度不能超过32 个有效字符序列，
字符集: 0~9，a~z，A~Z和下划线"_"

扩展标识符 由VHDL'93，VHDL2001支持，首尾用反斜杠"\"定界，
区分大小写，总与基本标识符不同，字符集：ASCII码，
反斜杠字符要双写，允许任意字符，包括保留字、类型字

▲ 保留字 类型字 专用字

保留字 预留用于专门用途的标识符，VHDL'87,VHDL'93和VHDL2001有差别

类型字 用于表示数据类型的标识符

专用字 用于表示特别信息和常量的标识符

第二部分 VHDL语言的学习基础(3)

•书写规定与基本句法单元

▲数及表示法

数制： 二进制、十进制、十六进制

书写格式： 被表示的数 ::= <基>#<用基表示的整数[.用基表示的整数]>#[<指数>]

<基>为2~16之间的十进制正整数,#号为定界符，<基>为10时可省略定界符和基。

<指数> ::= E[+]<十进制正整数> | E<–> <十进制正整数> ;

<用基表示的整数> ::= <扩展数字>{[下划线]<扩展数字>};

<扩展数字> ::= <数字> | <字符>

用字符A~F表示10~15的数字，不分大小写。

举例：

2#0001_0111_0010#　8#562#　16#172#　370　3.7E+2　--整数370的表示

2#0.100#　8#0.4#　16#0.8#　--实数0.5的表示

注意：在相邻数字之间插入下划线只为增加可读性，对数值无影响。

第二部分 VHDL语言的学习基础(4)

•书写规定与基本句法单元

▲字符、串、位串

字符： 用单引号括起来，例如，‘A’、‘a’、‘%’

串： 用双引号括起来。串内包含双引号字符时，用双写双引号来表示。串长度超过一行,用运算符“&”把两个子串连接起来,例如，

“A string”

“This string contains an″ ″ embedded string″ ″ in it”

“00001111zzzz”

“”

位串： 仅由0和1字符组成的串,用双引号括起来，前缀符号

- B 代表二进制
- O 代表八进制
- X 代表十六进制

例如，B“10” B“1111_0010_0001” --二进制位串

O“072” O“13” o“372” --八进制位串

X“FA” X“0d” x“FFE0” --十六进制位串

第二部分 VHDL语言的学习基础(5)

- 目标与分类
  - ▲信号、变量、常量和文件

从硬件的角度看，信号代表着实际电路中的某一连接线，而常量代表着实际电路中的电源或地。变量和文件没有与硬件直接的对应关系，通常它们只作为暂存和交换信息的载体使用

- 信号

抽象描述电路的导线，起保持改变的数值和连接子元件的作用。信号总是在元件的端口连接元件，元件间交换的信息仅通过信号传送

信号赋值不意味着立即更新其保持的原有内容，因为任何对信号的赋值操作只能作为预定数值存储在信号的驱动器中，仅当模拟时间经过起同步作用的语句或再一次启动了进程时才会发生更新动作。

允许利用属性存取过去和当前的数值，可以接受来自变量的赋值

第二部分 VHDL语言的学习基础(6)

- 目标与分类
  - ▲信号、变量、常量和文件
    - 信号

信号说明语句的书写格式为:

signal <信号名> {，…}: <类型字> [约束范围] [:= < 表达式>];

例如:

signal temp_sum: std_logic_vector (3 downto 0) := “0011”;

signal a, b: std_logic;

信号a和b取系统默认值，即：该类型的最左值或最小值 ‘0’。

信号代入符为“<=”,

例如:

output <= a xor b after 3 ns;

第二部分 VHDL语言的学习基础(7)

- 目标与分类
  - ▲信号、变量、常量和文件
    - 变量

用于描述硬件的高层次特性，在综合过程中可能推断为存储器件。

存在两种变量：规则变量和共享变量。

规则变量只能在子程序和进程语句中被说明和使用，是一个局部量。

共享变量的作用范围是全局的，即：跨越同级和向下的设计层次可见。

VHDL'87标准只支持规则变量，**VHDL'93**和**VHDL'2001**标准既支持规则变量也支持共享变量。

不允许在进程和子程序中说明共享变量。

变量说明语句的书写格式如下：

variable <变量名>{，<… >}：<类型字>[约束范围][:=<表达式>];

全局变量说明需加入保留字**shared**

第二部分 VHDL语言的学习基础(8)

- 目标与分类
  - ▲信号、变量、常量和文件
    - 变量

变量只在进程首次执行时初始化，并在进程被挂起和重新激活时保持原有数值不变。共享变量除赋值机制与信号不同外，其他类同于信号。

变量赋值语句的书写格式为：

[语句标号:] <变量名> := <表达式>;

例子：

m := 1;

counter := counter+1;

VHDL中对变量赋值是立即生效的。变量在赋值时不能用after引入附加延时

例如，下列赋值语句是错误的。

op1: counter := counter+1 after 2 ns;

VHDL'87标准不支持语句标号的使用

第二部分 VHDL语言的学习基础(9)

- 目标与分类

▲ 信号、变量、常量和文件

- 常量

代表具有语义的常数值。说明时赋值，一旦被赋值，就在整个程序执行中保持不变。

书写格式:

constant <常量名> {, ...}: <类型字> [:= <表达式>];

例子:

```
constant number_of_bytes : integer := 8;
constant number_of_bits : integer := 8*number_of_bytes;
constant m : integer := user_function(a,b);          -- user_function为函数
constant n : integer := a+b;
```

子程序中说明的常量，仅在子程序调用时有效，调用结束后它的值将被释放，常量说明语句中缺少赋值时称之为缓定常量。

第二部分 VHDL语言的学习基础(10)

- 目标与分类

▲ 信号、变量、常量和文件

- 文件

仅支持读写操作，不支持综合。不可以通过赋值来更新内容，只能通过子程序对文件进行读写操作。**STD**库**textio**包提供对文件的全部支持内容。

- 特殊目标

端口、类属**(generic)**、参量**(parameter)**及循环语句和生成语句的标号等。

端口所包含的目标为信号，作用是为实体内部数据与外部数据交换创建通道

类属的作用是为外部信息送入实体内部提供通道，类属传送的信息均为静态信息，即：常量。

参量的作用类似于**C**语言中的形式参数，参量在子程序中可以被说明为信号、变量或常量，而在函数中只能被说明为常量。

第二部分 VHDL语言的学习基础(11)

- 数据类型
  - ▲ 标量类型、复合类型、寻址类型、文件类型
    - 标量类型

      目标的数据值由单个元素构成
    - 复合类型

      目标的数据值由多个元素结组构成
    - 寻址类型

      指针类型，值指向存储空间，用于建立目标间的联系或控制存储空间
    - 文件类型

      外部文件存放的数据值序列的镜像，用于数据读入和写出
  - ▲ 预定义类型、用户自定义类型

第二部分 VHDL语言的学习基础(12)

•数据类型

▲ 预定义类型：STD库提供、可直接引用

- 整数类型(integer)

  位宽32位：值范围- 2^{31}+1~2^{31}-1，有符号数、无符号数、不能按bit操作

  子类型：自然数**natural**，正整数**positive**

- 浮点类型(real)：值范围 ± 1.0e+38、用于行为仿真
- 布尔类型(boolean)： 值范围 **true**、**false**
- 字符类型(character) **ASCII码：‘0’、‘1’、‘A’、‘@’**
- 字符串类型(string): **“0101”、“ABCDabcd”**
- 位值类型

  bit类型：**‘1’、‘0’**

  std_logic类型：**‘U’、‘X’、‘0’、‘1’、‘Z’、‘W’、‘L’、‘H’、‘-’**
- 位组类型

  bit_vector类型：**“1000_1010”**/std_logic_vector类型：**“zzzz_1001”**

第二部分 VHDL语言的学习基础(13)

- 数据类型
  - ▲预定义类型
    - 时间类型(time)

      单位：**fs、ps、ns、μs、ms、sec、min、hr**
    - 错误等级类型(severity)

      值状态：**note(注意)、warning(警告)、error(出错)、failure(失败)**
    - 寻址类型(line)**:** 指针类型，值指向存储空间
    - 文件类型(text)**:** 用于定义代表外部文件的目标
  - ▲用户自定义类型：非隐含定义，需要**type**语句
    - 整数类型(integer)、实数类型(real)、物理类型
    - 枚举类型(enumerated)**:** 离散值由左至右分配数值
    - 组类型(array)**:** 限定性组、非限定性组(元素规模暂不定)
    - 记录类型(record)**:** 不同类型数据集合，不能直接产生硬件对应
    - 寻址类型(access)和文件类型(file)

第二部分 VHDL语言的学习基础(14)

- 数据类型
  - ▲ 自定义语句格式
    - 标量类型定义

      type {< 类型名>} {, …} is [<类型字>] [约束范围];
    - 枚举类型定义

      type {< 类型名> } {, …} is ({<标识符> | <字符>}{, …}) ;
    - 限定性组类型定义

      type {< 组类型名>} {, …} is array ({<下标界> } {, …}) of <类型字>;

      <下标界>::= <类型字> |{<简单表达式> {to | downto}<简单表达式>}

      或者 <类型字> [range<简单表达式> {to | downto}<简单表达式>]
    - 非限定性组类型定义

      type {< 组类型名>} {, …} is array ({<类型字> range<>} {, …}) of <类型字>;

第二部分 VHDL语言的学习基础(15)

- 数据类型
  - ▲ 自定义语句格式
    - 物理类型定义

      type {<物理类型名>} {, …} **is** <约束范围>

      units <基本单位>;

      [{<导出单位>=<十进制数> | <以基表示的数> <单位名>;}]

      end units <物理类型名>;
    - 记录类型定义

      type {< 记录名> } {, …} **is record**

      {<元素名> {, …} : <类型字>; }

      end record < 记录类型名>;
    - 子类型定义

      subtype {<子类型名>} {, …} **is** <父类型字>[约束范围];
    - 组集合体

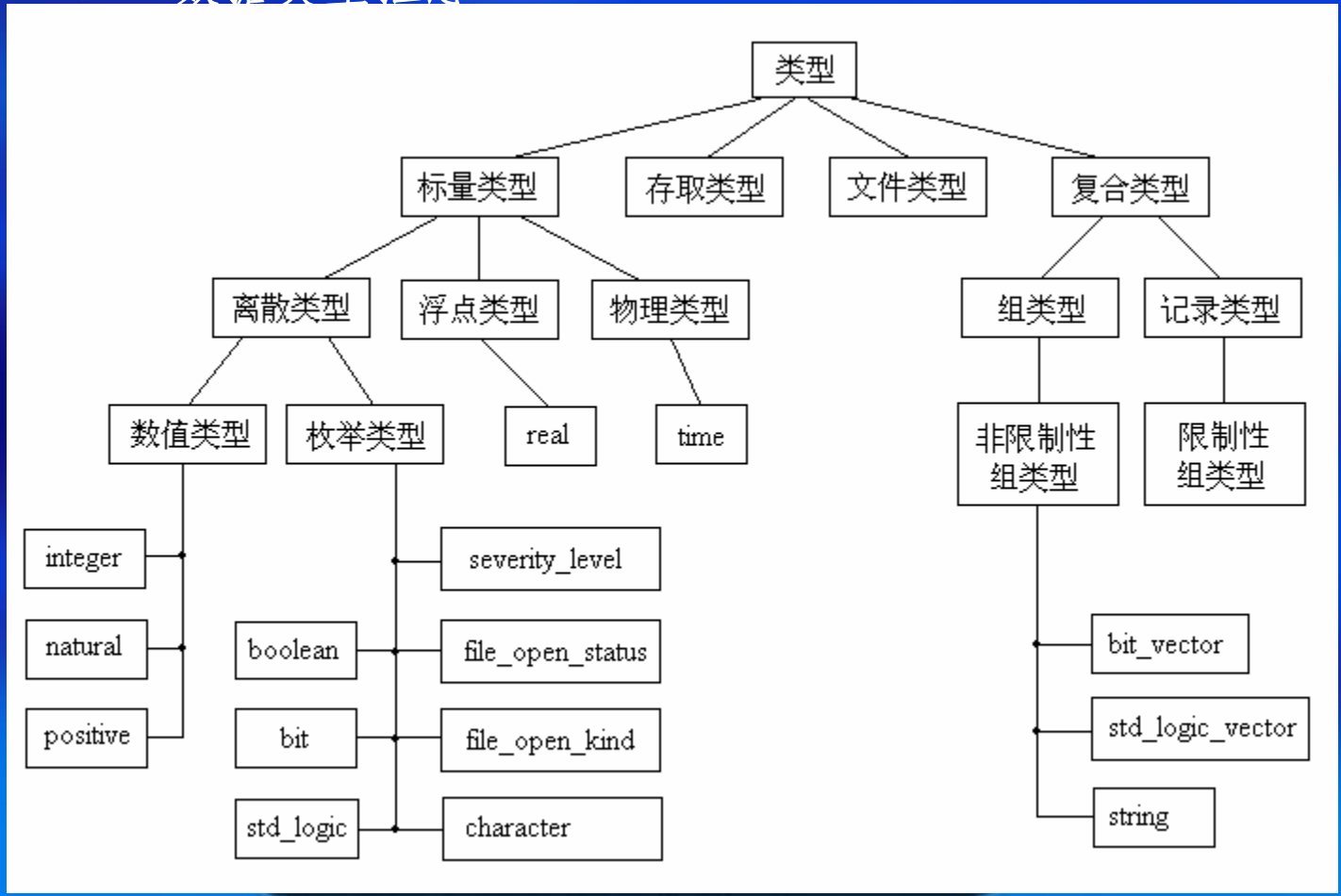
      ({[<选择列表>=>]<表达式>} {, …})

      命名结合、位置结合

第二部分 VHDL语言的学习基础(16)

- 数据类型

▲ VHDL数据类型汇总



第二部分 VHDL语言的学习基础(17)

•数据类型

▲ 标量类型属性

数值类、函数类、信号类、数据类型类、数据范围类，用于简化书写和获取标量类型的信息

- 数值类属性

T'left -- 能给出T的取值范围的左端值;

T'right -- 能给出T的取值范围的右端值;

T'low -- 能给出T的取值范围的低端值;

T'high -- 能给出T的取值范围的高端值;

T'ascending -- 如果T的取值范围按升序排列则为真，否则为假;

T'image(x) -- 能给出一个表达x的值的串;

T'value(s) -- 能给出由串s表达的T的值。

第二部分 VHDL语言的学习基础(18)

- 数据类型

▲ 标量类型属性

- 函数类属性

T'pos(x) -- 能给出x在T中的位置号;

T'val(n) -- 能给出在位置n处T的取值;

T'succ(x) -- 能给出x值所在位置的下一个位置处的值;

T'pred(x) --能给出x值所在位置的前一个位置处的值;

T'leftof(x) --能给出x值所在位置的左边位置处的值;

T'rightof(x) --能给出x值所在位置的右边位置处的值;

例如，**type day is (sun，mon，tue，wed，thu，fri，sat);**

Day'pos(sun) = 0 **Day'val(3) = wed**

Day'succ(thu) = fri **Day'pred(thu) = wed**

Day'leftof(mon) = sun **Day'right(mon) = tue**

第二部分 VHDL语言的学习基础(19)

- 数据类型

▲ 组类型属性

A'left(N) ——能给出A的第N维下标界的左边界值;

A'right(N) ——能给出A的第N维下标界的右边界值;

A'low(N) ——能给出A的第N维下标界的下边界值;

A'high(N) ——能给出A的第N维下标界的上边界值;

A'range(N) ——能给出A的第N维下标的取值范围;

A'reverse_range(N) ——能给出A的第N维下标取值范围的相反值;

A'length(N) ——能给出A的第N维下标界的长度;

A'ascending(N) ——如果A的第N维下标取值按升序排列则为真,否则为假;

第二部分 VHDL语言的学习基础(20)

- 表达式与运算符

▲ 表达式

由运算符和操作数组成

▲ 运算符

- 特殊运算符：**、abs、not
- 乘法运算符：*、/、mod、rem
- 正、负运算符：+、-
- 加减运算符：+、-、&
- 移位运算符：sll、srl、sla、sra、rol、ror
- 关系运算符：=、/=、<、>、<=、>=
- 逻辑运算符：and、or、nand、nor、xor、xnor

▲ 类型限定与类型转换

- 类型限定：显性表示数据的类型

语句格式：<类型字>’(数据)

第二部分 VHDL语言的学习基础(21)

- 表达式与运算符

▲ 类型转换

类型转换函数汇总表

函 数 名	功 能
▪ std_logic_1164 包集合	
to_stdlogicvector (a)	由 bit_vector 转换为 std_logic_vector
to_bitvector (a)	由 std_logic_vector 转换为 bit_vector
to_stdlogic (a)	由 bit 转换为 std_logic
to_bit(a)	由 std_logic 转换为 bit
▪ std_logic_arith 包集合	
conv_std_logic_vector (a，位长)	由 integer、unsigned 和 signed 转换为std_logic_vector
conv_integer (a)	由 unsigned 和 signed 转换为 integer
▪ std_logic_unsigned 包集合	
conv_integer (a)	由 std_logic_vector 转换为 integer

第三部分 VHDL顺序语句(1)

•顺序语句

建模进程、过程和函数功能的基本语句单元，仅用于进程、过程和函数中

▲ 变量赋值语句

[<标号>:] <变量名> := <表达式>;

注：(1)<表达式>可以包含信号、变量和常量

(2)赋值即刻生效

例：
```
signal sig：std_logic;                          --进程外部的信号sig
  …  …
event_on_sig: process is
    variable flag：boolean := false;            --说明变量flag并赋初值
begin
    flag := not flag;                           --变量赋值语句
    wait on sig;
end process  event_on_sig;
```

第三部分 VHDL顺序语句(2)

- 顺序语句
  - ▲ 简单信号赋值语句

[<标号>:] <信号名> <= [transport | inertial]<波形>;

注：(1) **<波形>::= {<表达式>[after<时间表达式>]} {, …}**

在并行语句区只为信号和常量，在顺序语句区也可为变量

(2)**transport**表示传输延时，起平移信号波形的作用，**inertial**表示惯性延时，延时由第一个**after**后的参数值决定

(3)赋值不能立刻生效，暂存在信号驱动器中，待同步事件发生时赋值到信号

例：

```
mux : process (a，b，sel) is                    --a、b和sel为敏感信号
begin
    case sel is
        when '0' =>  y <= a after 5 ns;          --若sel='0'，则a代入y
        when '1' =>  y <= b after 5 ns;          --若sel='1'，则b代入y
        when others => null;
    end case;
end process mux;
```

第三部分 VHDL顺序语句(3)

•顺序语句

▲ 等待语句

用于取代进程语句的敏感信号列表为进程提供同步，有四种形式:

[<标号>:]**wait** --无限等待

[<标号>:]**wait on** <信号名>{, …}; --直到信号活动或变化时结束等待

[<标号>:]**wait until** <布尔表达式>; --直到条件为真时结束等待

[<标号>:]**wait for** <时间表达式>; --直到延时时间到时结束等待

注：(1)<布尔表达式>至少要含有一个信号量，因进程一旦挂起，变量将不再改变，若要退出等待状态，只能靠信号的活动或变化引起布尔表达式的值为真

(2)允许混合使用

例： [<标号>:] **wait [on** <信号列表> **{, …}][until** <布尔表达式>**][for** <时间表达式>**];**

(3)等待语句与敏感信号列表不能同时使用

第三部分 VHDL顺序语句(4)

- 顺序语句

▲ **if** 语句： 按条件执行操作，用于表示**VHDL**模型的行为

[<标号>:] **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;

注：(1)<条件表达式>只能由关系运算符或逻辑运算符组成，给出布尔量

(2)**if**语句不允许对信号边沿做二选一 处理

(3)举例说明

例**3-2**中支持负整数运算的语句：

use IEEE.std_logic_signed**.all**; **use** IEEE.std_logic_arith**.all**;

例**3-3**中clk'event 用于返回信号事件属性

if clr = '1' **then** q <= '0';

elsif clk'event **and** clk = '1' **then** q <= d;

end if;

第三部分 VHDL顺序语句(5)

- 顺序语句

▲ **case语句：** 执行多选一操作，用于表示VHDL模型的行为

[<标号>:] **case** <选择表达式> **is**

{when <选择项> => <顺序语句>; **}**

end case [<标号>];

注: (1)<选择项> ::= <值表达式>|<离散值>|<值范围>|**others**

<值范围> ::= <值表达式> **{downto|to}** <值表达式>

(2)<选择项> 要覆盖所有可能的取值，不允许重复使用它们

(3)每次只执行一个分枝，**when** 子句的位置不影响执行结果

第三部分 VHDL顺序语句(6)

- 顺序语句

▲ 循环语句：用于描述具有重复结构或迭代运算的部分以简化程序代码，有三种形式：

- **loop语句**

[<标号>:] **loop**

{<顺序语句>;}

end loop [<标号>];

- **for loop语句**

[<标号>:] **for** <循环变量> **in** <离散范围> **loop**

{<顺序语句>;}

end loop [<标号>];

注：(1)<离散范围> ::= <简单表达式> **{to|downto}** <简单表达式>

(2)<循环变量> 为隐含定义、是整数、不允许修改

第三部分 VHDL顺序语句(7)

•顺序语句

▲ 循环语句：用于描述具有重复结构或迭代运算的部分以简化程序代码，有三种形式：

- **while loop语句**

[<标号>:]**while** <布尔表达式> **loop**

{<顺序语句>;}

end loop [<标号>];

注:(1)<布尔表达式>为真执行，否则结束

(2)循环变量非隐含定义

第三部分 VHDL顺序语句(8)

- 顺序语句

▲ **exit**语句：循环控制语句，可强迫循环从正常执行中跳到由语句标号所指定的新位置

exit [<语句标号>] [**when**<条件>];

注: 缺省<语句标号>或**when**子句则跳到循环语句**end loop**后面执行

▲ **next**语句：循环控制语句，可结束**next**语句后面的操作跳到由语句标号指定的环内新位置，只能实现内部循环控制

next [<语句标号>] [**when**<条件>];

例：
```
process (x, y) is
    variable eqi：std_ulogic;
begin
    eqi := '1';
    for i in x'range loop
        next when eqi = '0';   eqi := eqi and (x(i) xnor y(i))
    end loop ;
    eq <= eqi;
end process;
```

第三部分 VHDL顺序语句(9)

- 顺序语句

▲ **return**语句：起结束当前最内层过程体或函数体执行的作用，有两种形式：

当用于过程时，　[<语句标号>:] **return**;

当用于函数时，　[<语句标号>:] **return** <表达式>;

注: 保证函数体不执行到结尾的**end**并带回函数的返回值，返回值数据类型必须满足要求

▲ **null**语句：空操作，执行该语句无任何动作，只是把运行操作指向下一条语句。

[<语句标号>:] **null**;

▲ **assert**语句：断言语句来产生警告信息

assert <条件表达式> **[report** <报告信息>**][severity** <错误级别>**]**;

注: (1)<条件表达式>为真，执行下一条语句；否则输出<报告信息>和报告<错误级别>

(2)<报告信息>为文本说明信息。缺省**report**子句，默认消息为**"assertion violation"**

(3)<错误级别>为**failure**、**error**、**warning**和**note**。缺省**severity**子句默认为**error**

第三部分 VHDL顺序语句(10)

- 顺序语句

▲ **report**语句：类似于断言语句的功能，用于给出信息报告

[<语句标号>**:] report** <报告信息> **[severity** <错误级别>**];**

注:(1)仅有VHDL'93标准和VHDL'2001标准支持report语句。

(2)缺省**severity**子句时，**report**语句相当于**VHDL'87**标准中的一个条件为**false**、错误级别为**note**的**assert**语句

▲ 过程调用语句：相当于一条顺序语句或并行语句，在进程中使用为顺序语句。过程调用将启动对应过程体（子过程或函数）的执行

[<语句标号>**:]** <过程名> **[(**实参数列表**)];**

注: **(**实参数列表**)**是向过程体内部传递信息的接口。实参数可为信号、变量或常量

例：**proc_miltiplier (a，b，y);** --调用计算**y=a*b**的过程

第四部分 VHDL的模型结构(1)

- **设计实体：** VHDL的基本设计单元，由实体说明和构造体两部分组成

▲实体说明：用于描述一个设计的外部视图，即:定义实体名、界面接口、引入外部参数

```
entity <实体名> is
    [generic (<类属参数表>);]
    [port (<端口表>);]
    [<说明语句区>]
end entity [<实体名>];
```

注：(1)类属子句：提供用于规定端口大小、元件数目、定时的参数

{[constant]<参数>{, …}: [in]<类型名>[:=<简单表达式>]} {; …}

(2)端口子句：用于定义端口引脚名、数据流向、数据类型

{[signal]<端口名>{, …}: [<端口模式>]<类型名>[bus][:=<简单表达式>]} {; …}

✦ bus 用于表示具有三态特性的端口

✦ <端口模式> 用于表示端口信号的输入/输出方向：in、out、inout、buffer、linkage

✦ <简单表达式> 用于规定端口信号的缺省值，即：在非使用状态时为端口信号提供上拉电平。一旦端口信号被使用，则缺省值便自动被忽略

第四部分 VHDL的模型结构(2)

- 设计实体：VHDL的基本设计单元，由实体说明和构造体两部分组成

▲实体说明：

✦ <说明语句区>允许使用子程序和体、类型和子类型、常量或信号说明语句的区域

例：

```
library IEEE;
use IEEE.std_logic_1164.all;
entity ram is
    port(addr:  in std_logic_vector(15 downto 0);
         data:  out  std_logic_vector(31 downto 0);  enable :  in std_logic);
--类型说明和过程说明部分
    type instruction is array (1 to 8) of natural;
    type program is array (natural range<> ) of instruction;
    procedure initialization (signal content:  std_logic_vector(31 downto 0)) is
    begin
        content <= (others =>'0') after 5 ns;
    end procedure initialization;
end entity ram;
```

第四部分 VHDL的模型结构(3)

- 设计实体：VHDL的基本设计单元，由实体说明和构造体两部分组成

▲构造体：用于描述一个设计的内部视图，即：一个设计要实现的具体功能，实体说明与构造体必须结对使用，VHDL规定实体说明要放在构造体的前面，并总是先编译实体说明之后才能编译构造体，并把编译结果存放在当前设计库(WORK)中

```
architecture <构造体名> of <实体名> is
    [<说明语句区>]
begin
    {<并行语句区>}
end architecture [<构造体名>];
```

注：(1) <说明语句区>用于说明和定义内部信号、常量、数据类型、子过程、元件等

(2)内部说明信号不必规定信号方向

(3) <并行语句区>允许信号赋值、块、进程、子程序调用、生成、元件例示等语句

例：
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

第四部分 VHDL的模型结构(4)

- 设计实体：VHDL的基本设计单元，由实体说明和构造体两部分组成

例：

```
entity multiplier is
    generic(n: positive := 8);
    port(x, y: in std_logic_vector(n-1 downto 0); product: out  integer range 0 to 65535 );
end entity multiplier;
architecture behav of multiplier is
begin
    process is
        variable rs:   std_logic_vector(15 downto 0) := X"0000";
        variable rx:   std_logic_vector(15 downto 0);
    begin
        rx( 15 downto 0):= X"00"&x;    rs:=X"0000";
        for i in y'low to y'high loop
            if  y(i) = '1'  then  rs := rs + rx;   end if;
            rx(15 downto 0) := rx(14 downto 0)&'0';
        end loop;
        product <= conv_integer(rs);
    end process;
end architecture behav;
```

第四部分 VHDL的模型结构(5)

- 构造体功能的行为描述： 并行信号赋值语句、进程语句是行为描述的基本单元

▲并行信号赋值语句： 由简单信号赋值、条件信号赋值、选择信号赋值语句组成

- 简单信号赋值语句
- 条件信号赋值语句

[<语句标号>:] <信号名> <= [<延迟机制>]{<波形> **when** <条件表达式> **else**}

<波形> [**when** <条件表达式>];

注:(1)<波形> ::= {<简单表达式>[**after** <时间表达式>]}{, …}

(2)<条件表达式>和<波形>对信号敏感，一旦激活会按序检查由**when**所指定的条件，如果为真，则对应<波形>就代入信号，重复上述过程直至代入最后一个<波形>

(3)最后一个<波形>可以缺省条件，不允许把信号代入自身的描述

例： **architecture** behav **of** multiplexer **is**

begin

zmux: z <= d0 **when** sel = “00” **else**

d1 **when** sel = “01” **else**

d2 **when** sel = “10” **else**

d3 **when** sel = “11” ;

end architecture behav;

第四部分 VHDL的模型结构(6)

- 构造体功能的行为描述： 并行信号赋值语句、进程语句是行为描述的基本单元

▲并行信号赋值语句： 由简单信号赋值、条件信号赋值、选择信号赋值语句组成

- 选择信号赋值语句

[<语句标号>:] **with** <选择表达式> **select**

<信号名> <= [<延迟机制>]{<波形> **when** <选择项>, }

<波形> **when** <选择项>;

注:(1)<波形> 同条件信号赋值语句

(2)<选择项> ::= <值表达式> | <离散值> | <值范围> | **others**

<值范围> ::= <值表达式> **{downto | to}** <值表达式>

(3)对信号敏感，一旦激活会对所有的<选择项>进行检测不允许嵌套使用

(4)和case语句一样，既不允许<选择项>的取值被遗漏也不允许重复使用它们

(5)不允许把信号代入自身的描述

第四部分 VHDL的模型结构(7)

- 构造体功能的行为描述： 并行信号赋值语句、进程语句是行为描述的基本单元
  - ▲并行信号赋值语句： 由简单信号赋值、条件信号赋值、选择信号赋值语句组成
    - ● 选择信号赋值语句

例：
```
entity full_adder is
    port(a，b，c_in: in bit; sum，c_out: out bit);
end entity full_adder;
architecture truth_table of full_adder is
begin
    with bit_vector’(a，b，c_in) select                    --选择信号赋值语句
        (c_out，sum) <= bit_vector’(“00”) when “000”,
                        bit_vector’(“01”) when “001”,
                        bit_vector’(“01”) when “010”,
                        bit_vector’(“10”) when “011”,
                        bit_vector’(“01”) when “100”,
                        bit_vector’(“10”) when “101”,
                        bit_vector’(“10”) when “110”,
                        bit_vector’(“11”) when “111”;
end architecture behav;
```

第四部分 VHDL的模型结构(8)

- 构造体功能的行为描述

▲信号属性：用于得到有关信号事件和事项处理的历史信息

- 信号类属性

S'delayed(t) ——给出一个值与S相同，但被延迟t时间的信号

S'stable(t) ——给出一个boolean型信号，若至今S在t时间内没有事件发生，则值为真，否则为假

S'quiet(t) ——给出一个boolean型信号，若至今S在t时间内没有事项处理发生，即：静待t时间，则值为真，否则为假

S'transaction ——给出一个bit型信号，每当关于S的事项处理发生时，其值便发生相对前值的翻转

注：(1)省缺t时，取默认值0，并且生成的信号比S要延迟一个δ模拟时间

(2)不允许在过程和函数中应用

- 信号属性函数

S'event ——若在当前模拟周期内信号S有一个事件发生，则值为真，否则为假

S'active ——若在当前模拟周期内信号S有一个事项处理发生，则值为真，否则为假

第四部分 VHDL的模型结构(9)

- 构造体功能的行为描述

▲信号属性：用于得到有关信号事件和事项处理的历史信息

- 信号属性函数

S'last_event ——给出从信号S前一个事件发生至今所花费的时间

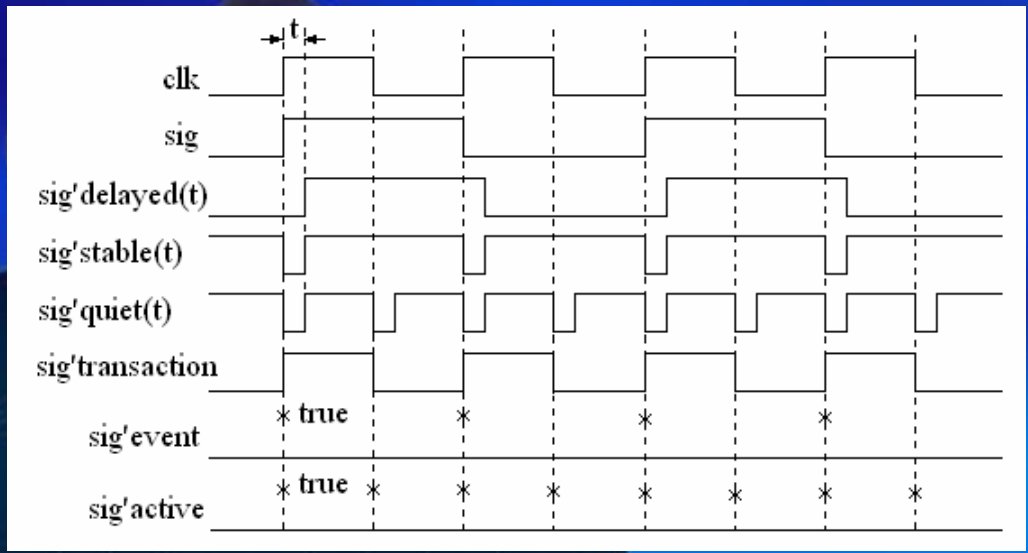
S'last_active ——给出从信号S前一个事项处理发生至今所花费的时间

S'last_value ——给出信号S在最近一个事件发生前那个时刻的值

注：(1)缺省t时，取默认值0，并且生成的信号比S要延迟一个δ模拟时间

(2)不允许在过程和函数中应用

(3)信号属性关系图例



第四部分 VHDL的模型结构(10)

- 构造体功能的行为描述

▲进程语句：由顺序语句组成，通过信号或端口实现信息传递

[<语句标号>:] process [(<敏感信号表>)] is

[<进程说明区>]

begin

{<顺序语句>; }

end process [<语句标号>];

注:(1)<敏感信号表>::= <信号名>{, <信号名>}

若缺省，则进程内必须包含wait语句，但不允许同时使用wait语句和敏感信号表

(2)<进程说明区>允许说明常量、变量、类型及子程序，不允许说明非共享变量

(3)变量只允许在进程内使用，进程外不可见。进程仅在首次执行时对变量初始化，之后不管进程挂起与否，变量总保持运算的中间结果而不会丢失。

(4)在模拟期间激活进程时，进程总是从第1条顺序语句(或wait语句后的顺序语句)开始执行，然后连续运行直至它上一次被挂起的地方为止

(5)例4－11说明

第四部分 VHDL的模型结构(11)

- 构造体功能的行为描述

▲无源进程： 由无源并行断言语句、无源并行过程调用语句、无源进程语句组成，用于监视实体的运行

```
entity <实体名> is
    [generic (<类属参数表>);]
    [port (<端口表>);]
    [<说明语句区>]
begin
    {<无源并行断言语句> | <无源并行过程调用语句> | <无源进程语句>}]
end entity [<实体名>];
```

注:(1)并行断言语句、并行过程调用语句、进程语句中无信号赋值操作就是无源的

(2)实体说明中包含的并行语句全局可见，不能以任何方式影响实体运行

▲并行assert语句： 用于检测行为建模中的错误，与串行assert语句格式相同

第四部分 VHDL的模型结构(12)

- 构造体功能的子结构描述：块、子程序、元件是结构描述的基本单元

▲块语句：能把多条并行语句聚合在一起的并行语句，用于描述功能相对比较独立的子结构

```
[<标号>:] block [(<卫士表达式>)] [is]
    [generic (<类属参数表>);
    [generic map(<类属参数表>);]]
    [port (<端口表>);
    [port map(<端口表>);]]
    [<说明语句区>]
begin
    {[guarded] <并行语句>;}
end block [<标号>];
```

注:(1)**guarded**代表由<卫士表达式>隐含说明的一个布尔型信号。<卫士表达式> 用于控制以**guarded**做前缀的并行信号赋值语句的运行。若值为真，**guarded**就接通被它保护的并行信号赋值语句的驱动器允许赋值操作；否则就切断驱动器禁止赋值操作。没有**guarded**前缀的并行信号赋值语句将不受影响

(2)类属和端口子句用于为块语句定义界面，作用与实体说明中的情况相类似。

第四部分 VHDL的模型结构(13)

- 构造体功能的子结构描述：块、子程序、元件是结构描述的基本单元

▲块语句

(3)<说明语句区>允许使用常量、类型、子类型、信号和子程序等说明语句。说明项目只在块内有效，块外不可见。块外说明的信号、类属参数和端口信号，在块内可用，当块内有同名说明项目时，块内说明项目占优

(4)允许在块语句内部使用块语句，即：允许对一个系统进行块嵌套描述。但块嵌套不一定使设计结构变得清晰。实际上，块嵌套主要用于配置语句，注意构造体本身也等价于一个块结构。

例：
```
library IEEE; use IEEE.std_logic_1164.all;
entity mux_2 is
    generic (sig_width: positive :=16);
    port (x0，x1: in std_logic_vector(0 to sig_width-1); sel: in std_logic;
          y: out std_logic_vector(0 to sig_width-1));
end entity mux_2;
architecture behav of mux_2 is
begin
```

第四部分 VHDL的模型结构(14)

- 构造体功能的子结构描述：块、子程序、元件是结构描述的基本单元

▲块语句

```
b1: block is
    generic(width:  positive);
    generic map(width => sig_width);
    port(d0，d1:  in std_logic_vector(0 to width-1);  sel:  in std_logic;
         result:  out std_logic_vector(0 to width-1));
    port map(d0 => x0，d1 => x1，result => y，sel => sel);
    signal temp0，temp1:  std_logic_vector(0 to width-1);
begin
    temp0 <= d0 when sel = '0' else (others => '0');
    temp1 <= d1 when sel = '1' else (others => '0');
    b2: block is
    begin
        result <= temp0 or temp1;
    end block b2;
end block b1;
end architecture behav;
```

第四部分 VHDL的模型结构(15)

- 构造体功能的子结构描述：块、子程序、元件是结构描述的基本单元

▲子程序：由过程和函数组成，用于实现重复结构的功能和行为。过程通过接口参数表与外部直接交换信息，函数通过接口参数表接受外部信息、用函数调用的返回值回馈信息给外部。主要差别：过程调用相当于一条语句，函数调用只返回一个值

● 过程

procedure <过程名> **[**(<形式参数表>)**]** **is**
　　{<过程说明区>}
begin
　　{<顺序语句>；}
end procedure [<过程名>];

注:(1)<形式参数表> ::= {[constant|variable|signal]{<参数>}{, …}: [<方向模式>]
<子类型>[:= <简单表达式>]} {； …}

(2) <方向模式>仅有in、out和inout三种。若缺省<子类型>，则in方向的参数默认为常量，inout或out方向的参数默认为变量。缺省<方向模式>时，默认方向为in

(3)允许类型、子类型、变量、常量和子程序说明，但不允许说明信号

(4)每次调用过程都进行初始化，调用结束后便释放内部变量，调用时，若缺省实参或不连接(用open取代实参的位置)，则以<简单表达式>作为缺省初始值

第四部分 VHDL的模型结构(16)

- 构造体功能的子结构描述：块、子程序、元件是结构描述的基本单元
  - 过程

注:(5)信号在过程内总是可见的，可省缺<形式参数表>在过程内直接寻访外部信号。但若要在调用过程的进程内驱动这些信号，还必须在进程内说明该过程才能做到。为避免错误地修改全局信号，不推荐用非<形式参数表>内的参数来传递信息

(6)在含有敏感信号表的进程中调用过程时，过程内部禁用**wait**语句

- 函数：可以被看作是一个能和运算符一起参加运算的广义表达式，其功能可以通过一组顺序语句来实现

[pure|impure] function <函数名> **[(**<形式参数表>**)] return** <类型名> **is**
  {<函数说明区>}
begin
  {<顺序语句>; }
end [function] [<过程名>];

注:(1)函数只允许形式参数为常量和信号，不允许为变量，缺省<类型字>默认为常量;每个形式参数的方向模式都必须为**in**，缺省方向模式字默认为**in**方向

(2)信号在函数内总是可见的，在函数内部可直接寻访外部信号。但不允许在其内部使用信号赋值语句

第四部分 VHDL的模型结构(17)

- **构造体功能的子结构描述：** 块、子程序、元件是结构描述的基本单元
  - 函数：

注:(3)函数不能作为一条语句使用，只能返回一个用于表达式计算的值，其数据类型由保留字**return**后面的<类型名>来规定，因函数总使用**return**语句返回计算值给调用者，实际上最后一条语句就是**return**语句

(4)函数允许进行类型、子类型、变量、常量和子程序说明，但不允许说明信号

(5)**pure**或**impure**用于指定纯函数或非纯函数，用相同形参调用函数总能返回一个相同值为纯函数；可能会返回不同值为非纯函数。缺省保留字**pure**或**impure**时，函数被默认为纯函数

- 过程与函数的调用：既允许在顺序语句区，也允许在并行语句区调用

[<语句标号>:] <子程序名> **[(<参数互连表>)];**

注:(1)<参数互连表>::={**[**<参数名> =>**]**<表达式>|<信号名>|<变量名>|**open}{**，…**}**

(2)参数互连形式：命名结合，用符号“=>”显性地指明互连关系，书写顺序不重要；位置结合，用实参占形参位置来相连，书写顺序非常重要。允许两种方法混合使用。互连参数有输入、输出方向之分

第四部分 VHDL的模型结构(18)

- 构造体功能的子结构描述：块、子程序、元件是结构描述的基本单元

▲子程序：

- 子程序重载：指多个子程序使用相同名字的情况，VHDL编译工具通过检查子程序参数是否具有相同数据类型来区分重名的子程序，例如：
  - ✦ 子程序调用中出现的参数数目
  - ✦ 子程序调用中出现的参数类型
  - ✦ 子程序调用为实现参数结合所用的参数名字
  - ✦ 在函数调用下返回值的数据类型

注:(1)如果使用的重载子程序使VHDL无法区分上述因素，那么就会产生错误。

(2)在VHDL中，运算符作为一种预定义的函数也可以重载

第四部分 VHDL的模型结构(19)

•构造体功能的子结构描述：块、子程序、元件是结构描述的基本单元

▲元件语句：用于描述设计的层次结构，由元件说明和元件例示两部分组成。元件说明仅指定元件的外部界面；元件例示用于完成元件与外部信号间的互连，不受设计层次限制。元件只与相关的实体接口连接，不依赖于任何库单元，元件语句本身不能赋予元件功能，为使元件获得指定的功能需借助配置语句来完成

•元件说明语句：只能在包集合、构造体、块中的说明区使用

component <元件名> **[is]**
  [generic(<类属参数表>**);** **]**
  [port (<端口表>**);** **]**
end component [<元件名>**];**

注:(1)**generic**子句用于定义引入元件内部的常量参数，**port**子句用于规定元件的输入和输出端口信号。<类属参数表>和<端口表>的书写格式与实体说明语句相同

(2)元件说明与实体说明表达的概念不同：一个实体说明最终可能代表一个实际的集成电路器件/一个放在硅芯片上的标准设计模块，是一个分离的设计单元，可以独立地进行分析，结果要放入设计库中。而一个元件说明只是定义了一个包含在构造体内的虚拟模块

第四部分 VHDL的模型结构(20)

- 构造体功能的子结构描述： 块、子程序、元件是结构描述的基本单元

▲元件语句：

●元件例示语句：

<例示标号>: **[component]** <元件名>

[generic map(<类属参数互连表>**)]**

[port map(<端口互连表>**)];**

注:(1)<例示标号>必须是唯一的。

(2)实际参数与类属参数、实际信号与端口信号之间的互连可采用位置结合或命名结合方法来完成。

(3)**VHDL'87**标准不支持使用保留字**component**。**VHDL'93**和**VHDL'2001**标准使用该保留字主要用于区分一个例示是元件例示还是直接实体例示

例： **library** IEEE;

use IEEE.std_logic_1164.**all**;

use IEEE.std_logic_arith.**all**;

use IEEE.std_logic_unsigned.**all**;

use WORK.divide_unit;

第四部分 VHDL的模型结构(21)

- 构造体功能的子结构描述：块、子程序、元件是结构描述的基本单元
  - ▲元件语句：

```
entity divider is
    generic(m: integer := 12);
    port (clk:  in std_logic;  x，y:  in std_logic_vector(m-1 downto 0);
          z:  out std_logic_vector(m-1 downto 0));
end entity divider;
architecture behav of divider is
    component divide_unit is                                    --元件说明
        generic(m:  integer);
        port(clk:  in std_logic;  rd:  out std_logic;
             operand_left，operand_right: in std_logic_vector(m-1 downto 0);
             result:  out std_logic_vector(m-1 downto 0));
    end component divide_unit;
    signal rd:  std_logic;
 begin
    div: component divide_unit                                  --元件例示
    generic map(m)
    port map(clk，rd，x，y，z);
end architecture behav;
```

第四部分 VHDL的模型结构(22)

•**配置：** 分默认连接、配置指定、配置说明和直接例示4种形式，可用于为例示元件赋予功能或为实体说明选配构造体。是初级设计单元，可单独地编译，编译后并入设计库中

▲默认连接：指将一个元件与当前工作库中的同名实体相连接的情况。默认连接的元件不能被连接的实体包含。默认连接只在被连接的实体可见时才发生。使被连接的实体透明需使用**use**子句

例**4-20**中用 **use** WORK.divide_unit;

使被连接的实体可见

注：默认连接总把连接直接指向当前工作库中最新编译的与同名实体说明对应的构造体

▲配置指定：用于建立元件例示与其他实体之间的关系，同时完成把实体功能赋予元件的任务。只能在构造体、块或生成语句的说明区内使用

for <元件标识> **use entity** <库名>.<实体名>**[**(构造体名)**]**

[generic map(<类属参数互连表>**)]**

[port map(<端口互连表>**)];**

注:(1)<元件标识> ::= **{**<例示标号>**}{**，…**}** | **others** | **all**: <元件名>

若取**others**或**all**保留字时，则分别代表所有剩余例示元件或全部例示元件。

(2)缺省构造体名时，连接指向最新编译的构造体

第四部分 VHDL的模型结构(23)

- 配置:

▲配置指定:

注:(3)<类属参数互连表>和<端口互连表>与元件例示语句书写相同。

(4)构造体中的配置指定对**block**语句和**generate**语句中的元件例示无效。要为块或生成语句内的例示元件赋予功能，必须把配置指定放在**block**语句或**generate**语句的说明区内，**VHDL'87**标准没有为**generate**语句提供说明区，故须采取在生成语句中加入块语句的方法来解决。

```
例: architecture behav of divider is
        component divide_unit is                                    --元件说明
            generic(m:  integer);
            port(clk:  in std_logic;  rd:  out std_logic;
                    operand_left，operand_right: in std_logic_vector(m-1 downto 0);
                                        result:  out std_logic_vector(m-1 downto 0));
        end component divide_unit;
    signal rd:  std_logic;
    for div: divider_unit use entity WORK.divide_unit(behavior);          --配置指定
    begin
    div: component divide_unit  generic map(m) port map(clk，rd，x，y，z);
    end architecture behav;
```

第四部分 VHDL的模型结构(24)

- 配置:

▲配置说明：实现从元件例示到设计实体连接或完成实体与构造体选配的最基本的语句

configuration <配置名> **of** <实体名> **is**

<配置块>

end configuration [<配置名>];

注:(1)<配置块> ::= **for** <构造体名>

[{**for** {<例示标号>{, …}|**others**|**all**: <元件名>

<捆绑指定>

[**generic map**(<类属参数互连表>)]

[**port map**(<端口互连表>)];

[<配置块>]

end for; }]

end for;

(2)<捆绑指定> ::= **use entity** <库名>. <实体名>[(构造体名)]

|**configuration** <库名>. <配置名>

(3)允许递归使用<配置块>，允许指定元件与实体的端口、类属参数间的显性连接。

(4)<配置块>中无配置信息时，把设计实体与构造体进行捆绑

第四部分 VHDL的模型结构(25)

- 配置:

▲直接例示: 是一种隐含配置的元件例示语句。不需要元件说明直接指向实体对端口例示。

直接例示中隐含的配置不是直接指向实体就是直接指向实体的配置

<例示标号>: **use entity** <库名>.<实体名>**[(**构造体名**)] | configuration**<库名>.<配置名>

[generic map(<类属参数互连表>**)]**

[port map(<端口互连表>**)];**

例:

```
entity logic_block is
    generic(n: time := 5 ns);
    port(x, y: in bit; z: out bit);
end entity logic_block;
architecture behav of logic_block is
    … …
begin
    gate1: use entity WORK.or3(behavior)                    --直接例示
           generic map(ipd => n, opd => n)
           port map(a => x, b => y, c => '0', z => z);
    … …
end architecture behav;
```

第四部分 VHDL的模型结构(26)

- 配置:
  - ▲直接例示:

当前工作库(WORK)中的三输入或门为:

```
entity or3 is
    generic(ipd，opd:  time);
    port(a，b，c:  in bit;  z:  out bit);
end entity or3;
architecture behavior of or3 is
begin
    z <= a or b or c;
    …    …
end architecture behavior;
```

第四部分 VHDL的模型结构(27)

•包集合与设计库

▲包集合：用于集中组织服务于公共用途的有关说明和定义，使数据类型、常量、信号、元件及子程序全局可见，以便在所有设计单元中可以不加说明的引用它们，实现简化源代码、增强设计可重用性和程序可读性的目的。由包集合说明和包集合体(package body)两部分组成。

•包集合说明：主要用于定义包集合的界面，即：定义被说明项目对外可见的部分。不允许定义实体/构造体对，因此，包集合本身不能直接表达任何一个电路。

package <包集合名> **is**

<包集合说明项目>

end package [<包集合名>];

注:(1)<包集合说明项目>由类型、子类型、常量、信号、元件和子程序说明语句所组成

(2)有两类包集合，一类是VHDL预定义的标准包集合，主要用于定义VHDL的类型、运算符和转换函数等；而另一类是用户定义的包集合，主要用于定义描述一个实际设计所需要的一系列说明、元件或子程序。

(3)包集合实际上是一个库单元，若干个由用户编写的包集合汇合在一起可以组成一个设计库

第四部分 VHDL的模型结构(28)

- 包集合与设计库
  - 包集合说明:

注:(4)包集合说明是一个主设计单元，可以独立地编译并插入到当前工作库(WORK)中

(5)引用包集合内部的说明项目，需要使用use子句并通过选择项目名来使对应的说明项目透明可见，例如

use {<库名>.<包集合名>.<选择项目名>}{，…};

(6)<选择项目名> ::= <说明项目名> | **all**

引用保留字**all**会使包集合说明中的全部说明项目均对外透明可见。

(7)把用户编写的通用包集合放入非当前工作库，需参考VHDL工具商提供的用户手册

例：
```
package display_constants is
    constant addr_width : positive :=9;
    constant num_words : string :="296";
    constant data_width : positive :=8;
    constant roll_speed : string :="85";
end package display_constants;
library IEEE;   use IEEE.std_logic_1164.all;
use WORK.display_constants.all;
```

第四部分 VHDL的模型结构(29)

•包集合与设计库

•包集合体：是一个可选项，当包集合说明含有缓定常量或子程序说明时，就需要有一个对应的包集合体来为缓定常量赋值或给出子程序所缺少的具体细节(功能)

package body <包集合名> **is**

<说明项目细节>

end package body [<包集合名>];

注:(1)<包集合名>必须与包集合说明中的相同

(2)<说明项目细节>中，允许使用额外的类型、子类型、常量和子程序说明，但除了缓定常量或子程序说明外，其它说明不允许与包集合说明中的项目重复，并且在包集合体内不允许使用信号说明语句。此外，除非是缓定常量或子程序，否则包集合体内的说明项目对外都是不可见的

(3)包集合体是一个次级设计单元，在其对应的主设计单元被编译和插入**WORK**库后独立地被编译和插入到同一**WORK**库中

(4)见例**4-35**

第四部分 VHDL的模型结构(30)

•包集合与设计库

▲设计库： 由被编译的包集合、实体/构造体对和配置的数据集合所组成。设计库中存放的设计数据可作为其他VHDL设计单元的共享资源。设计库对设计单元并不都是透明的，一个设计单元要访问某个设计库，那么就要把有关该库的说明语句放在这个设计单元的最前面

•库语句：

library {<库名>}{，…};

注:(1)库说明语句相当于一个路径指针，总指向存放库数据的与库同名的文件夹

(2)VHDL语言支持多个设计库同时并存，但库与库之间是独立的，不能相互嵌套

(3)设计库可划分为STD库、WORK库、IEEE库、VITAL 库、资源库和用户库

第四部分 VHDL的模型结构(31)

•包集合与设计库

- **STD库**：含有**standard**和**textio**两个包集合。**standard**包集合主要支持标准**VHDL**的预定义类型和基本运算符的使用，例如，**bit**和**bit_vector**类型等。**VHDL**中隐含存在

  library STD;

  use STD.standard.**all;**

  因此，使用**standard**包集合无须再进行说明。

  textio包集合主要支持对文件类型数据的读写操作，但不支持对其进行赋值操作和逻辑综合。它实际上是专为**VHDL**模拟工具提供的与外部计算机文件管理系统交换数据的一个界面，使用前需要用**use**子句。

- **WORK库**：是**VHDL**指定的当前工作库，**VHDL**把所有的当前设计都存放在该库之中，它总是透明的，这相当于**VHDL**中隐含存在着下列形式的语句

  library WORK;

  因**VHDL**工具供应商的不同，**WORK**库实际指向的物理路径可能不与**WORK**库同名。例如，**Altera**公司提供的**MAX+PLUS II VHDL**工具就以**max2work**作为当前工作库映射的实际物理路径。

第四部分 VHDL的模型结构(32)

- 包集合与设计库
  - IEEE库：主要由标准包集合std_logic_1164、numeric_bit和numeric_std、math_real和math_complex组成。

    std_logic_1164标准包集合支持建立在std_logic和std_logic_vector类型基础上的标量类型、复合类型和基本运算符的使用、numeric_bit和numeric_std包集合支持有符号数和无符号数的运算，均可逻辑综合；而math_real和math_complex包集合仅支持实数和复数运算，不能综合成为硬件。

    Synopsys公司的std_logic_arith、std_logic_unsigned和std_logic_signed三个包集合也包括在IEEE库中，作用与IEEE标准包集合numeric_bit和numeric_std基本相同

    IEEE库非隐含可见，使用前必须要用库说明语句和use子句使其透明

  - VITAL库：是符合IEEE Standard 1076.4标准的IEEE库。由含有精确的ASIC时序模型的时序包集合vital_timing和基本元件包集合vital_primitives所组成，支持以ASIC单元的真实时序数据对一个VHDL设计进行精细地模拟验证，可以大大地提高VHDL门级时序模拟的精度

第四部分 VHDL的模型结构(33)

•包集合与设计库

- •资源库：厂商库，例如，**Synopsys**公司的**synopsys**库和**Altera**公司的**lpm**库。 库中存放着的与逻辑门一一对应的实体，使用前必须要用库说明语句对其进行说明

- •用户库：指由用户提供的公用包集合和实体/构造体对等设计数据汇集在一起所组成的库，主要用于提高设计效率和实现设计重用。用户库不是隐含可见的，使用前需要用库说明语句对其进行说明

例： **Altera**公司的资源库： **altera** 、 **altera_mf**和**lpm**资源库

库	包集合	文件	功能	设计工具
altera	maxplus2	maxplus2.vhd	提供原语、兆核函数、宏核函数模型	Max+plusII
	megacore	megacore.vhd	提供兆核兆函数模型	QuartusII
altera_mf	altera_mf_components	altera_mf_components.vhd	提供宏核函数模型	QuartusII
lpm	lpm_components	lpm_pack.vhd	提供参数化模型库函数	Max+plusII QuartusII

第五部分 深入理解VHDL(1)

•决断信号： 指用判决函数对连接至一个多源驱动信号的多个驱动器的输出值判决所获得的一个起支配性作用的输出值的信号

▲多源驱动问题

●单源驱动器 **x <= a after 5 ns;**

●多源驱动器 **y <= a after 5 ns;**
y <= b after 5 ns;

注:(1)信号同时受到多个驱动器输出值的驱动

(2)必须确定多源驱动信号最终由哪个驱动器的值所驱动

▲决断信号： 由多个驱动源和一个判决函数组成，

signal <决断信号名> **{, …}**: <判决函数名> <类型字> **[**范围**] [**:= <表达式>**];**

注:(1)与常规信号说明语句差别仅在于要求在类型字前加一个判决函数名作前缀

(2)<表达式>提供驱动器的初始化值，缺省取默认值，信号初始值由判决函数裁定

(3)可先定义决断子类型，再定义所需要的决断信号

subtype <决断子类型名> **{, …} is** <判决函数名> <类型字> **[**范围**];**

例： **subtype** resovled_wired_and **is** wired_and std_ulogic;
signal result: resovled_wired_and;

(4)一旦待决断信号活跃，判决函数就会被隐含地调用，用户不能控制该函数调用

第五部分 深入理解VHDL(2)

- **决断信号：** 用判决函数对连接至一个多源驱动信号的多个驱动器的输出值判决所获得的一个起支配性作用的输出值的信号

▲判决函数：用于定义决断类型、决断信号和确定多源驱动发生时哪一个驱动源的输出可作为待决断信号的最终值

注:(1)输入必须是与决断信号类型相同的一维非限制性组，返回值必须与决断信号类型相同，判决函数不能对多源驱动信号到达的顺序提出限制性要求

(2)判决函数被隐含调用时，一个由多源驱动信号输出值按顺序组成的一维组就会以实参数的形式传递到判决函数的内部，判决函数总是基于这些数据做出仲裁以确定决断信号的最终输出值

(3)判决函数必须是一个纯函数，以保证对每一组给定的多源驱动值总能返回一个可预测的决断信号的更新值

(4)包集合**std_logic_1164**定义了一个判决函数**resolved**，主要用于定义标准位值决断类型**std_logic**和对多源驱动信号进行仲裁

(5)见例**6-3**、 **6-4**、 **6-5**

▲决断端口：指具有决断子类型的端口，被多个进程驱动或有多个元件例示与之连接时需要

▲驱动值属性：一种能读取前缀信号值的属性，属性字为**driving_value**

s'driving_value

第五部分　深入理解VHDL(3)

•生成语句：是一个内部包含有多个待生成的并行结构或行为的并行语句。有两种生成结构，即：迭代结构和条件结构

▲迭代生成语句：　可根据需要生成多条描述规则结构或重复性行为的并行语句

<标号>: **for** <循环变量> **in** <离散范围> **generate**

[[<说明语句区>**]**

begin]

{<并行语句区>**}**

end generate [<标号>**];**

注:(1)<标号>用于区分生成的结构，<离散范围>与**for loop**语句中的含义相同，对每一个离散值，<说明语句区>和<并行语句区>都要被执行一次

(2)由<循环变量>给出的每一个值均被称为生成参数，生成参数与具有离散范围父类型的常量相类似

(3)当生成语句不包括<说明语句区>时，保留字**begin**可以省略

第五部分　深入理解VHDL(4)

•生成语句

▲条件生成语句：可生成描述边界不规则而内部具有规则结构或重复行为的并行语句

<标号>：**if** <条件表达式> **generate**
　　[[<说明语句区>**]**
begin]
　　{<并行语句区>**}**
end generate [<标号>**];**

注:(1)<条件表达式>用于控制并行语句的复制与否，若条件为真，则生成语句的<说明语句区>和<并行语句区>就被包含在设计中，反之，就被在设计中删除。

(2)<标号>用于区分条件为真时生成的结构

▲生成语句的配置：对生成语句内的元件例示可使用默认配置和扩展配置说明。扩展配置说明比常规配置说明在配置块中增加了<生成语句标号>及附加生成参数以便挑选迭代生成结构中特定的元件例示进行配置

格式见P167页

注:(1)对迭代生成语句时，<生成语句标号>后面的可选项仅用于从迭代生成结构中挑选特定的元件例示，或者选择一组元件例示

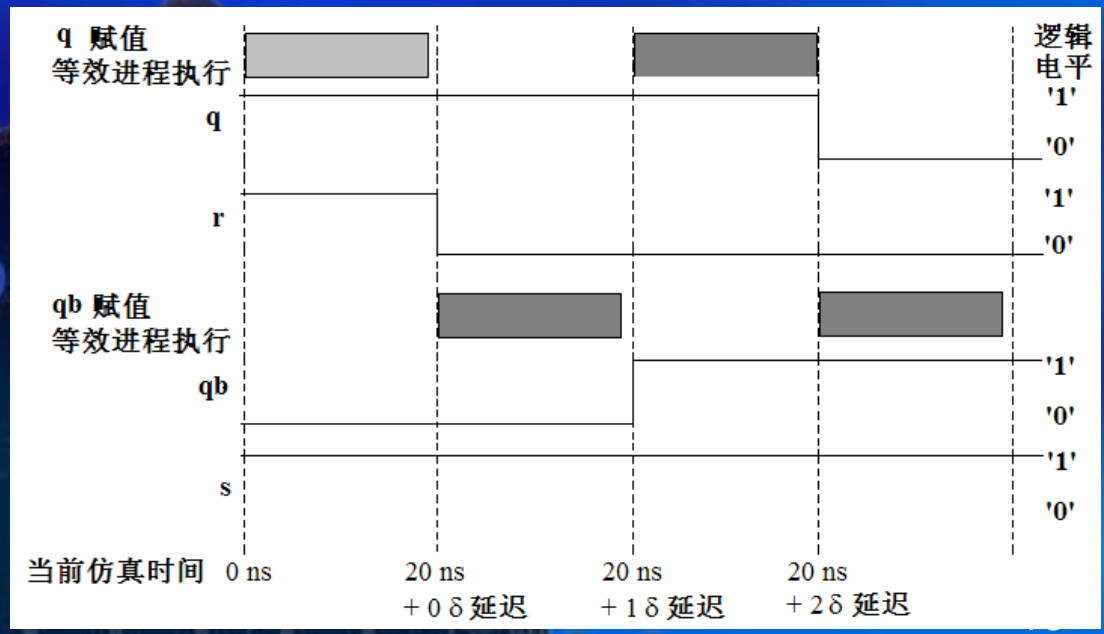
(2)对条件生成语句，仅用不带可选项的<生成语句标号>。如生成语句创建元件例示，配置就捆绑实体至元件例示；若没有创建元件例示，配置就被忽略

第五部分 深入理解VHDL(5)

•信号延迟的描述

▲δ延迟的概念：为保证模拟过程不因并行语句的执行顺序不同而影响模拟结果，VHDL对所有0延时信号赋值所引发的事项处理都插入了一个无限小的时间延迟量δ，以消除模拟结果对进程或等效进程执行顺序的依赖，这被称为δ延迟机制。用δ表示时间延迟量意味着任何有限多个δ延迟的时间累积值都将小于VHDL系统中的一个最小时间单位，即：1 fs

例：
```
architecture delta_delay of RS_flipflop is
begin
    q <= s nand qb;
    qb <= r nand q;
end architecture delta_delay;
```



第五部分　深入理解VHDL(6)

•信号延迟的描述

▲模拟周期：VHDL的仿真由初始化及重复执行进程组成，每次对进程的重复执行组成一个周期。在每一个周期中，所有的信号值均要被执行和计算。若计算结果有某个事件发生，就将启动与之敏感的对应进程，并作为该模拟周期的一部分被执行

模拟周期的三个主要阶段：

- 进程被激活
- 更新信号值
- 执行被激活的进程直至被挂起

注：在VHDL模拟期间，每一个模拟时刻都可能被插入若干个δ延迟

▲延缓进程：在所有的一般(非延缓)进程都被执行并被悬挂之后才被激活的进程

```
[<进程标号>: ] [postponed] process [({<信号名>}{，…})] [is]
    {<进程说明区>}
begin
    {<顺序语句区>}
end [postponed] process [<进程标号>];
```