VHDL语言详解

主讲：张晓磊

内容提要

- VHLD概述；
- VHLD的一些基本概念；
- VHDL的数据对象，数据类型及类型转换，运算符等；
- VHDL的顺序描述语句；
- VHDL的并行描述语句；
- VHDL的子程序结构；
- VHDL库、程序包和配置；
- VHDL的预定义属性；
- VHDL的重载；
- VHDL结构体的描述方式。

概述（1/4）

- Very high speed integrated Hardware Description Language (VHDL)
  - 发展史
    - 1980年，美国国防部的VHSIC（Very High Speed Integrated Circuit）计划；
    - 1982年，正式诞生“VHSIC硬件描述语言”；
    - 1987年12月，VHDL被接纳为IEEE 1076标准；
    - 1993年，该标准被修订，更新为新的VHDL标准IEEE 1164；

概述（2/4）

- 用语言的方式而非图形等方式描述硬件电路
  - 容易修改
  - 容易保存
- 特别适合于设计的电路有：
  - 复杂组合逻辑电路，如：
    - 译码器、编码器、加减法器、多路选择器、地址译码器……
  - 状态机
  - 等等……

概述（3/4）

- 另一种硬件描述语言Verilog HDL
  - 1983到84年，Phil Moore在Gateway Design Automation发明；
  - 1989年，被Candence收购；
  - 1995年12月，通过IEEE标准(IEEE 1364)；
  - 语法上Verilog基于C语言，简单易学；

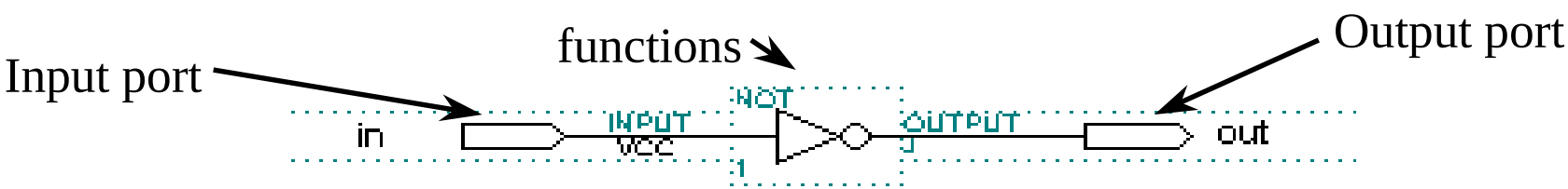
概述（4/4）

- VHDL vs. Verilog HDL
  - USA—IBM,TI,AT&T,INTEL…VHDL;
  - USA—Silicon Valley…Verilog;
  - Europe—VHDL;
  - Japan—Verilog;
  - Korea—70-80%VHDL;

VHDL的一些基本概念

基本概念（1/7）

- VHDL的功能
  - 描述输入端口/输出端口
  - 描述电路的行为和功能

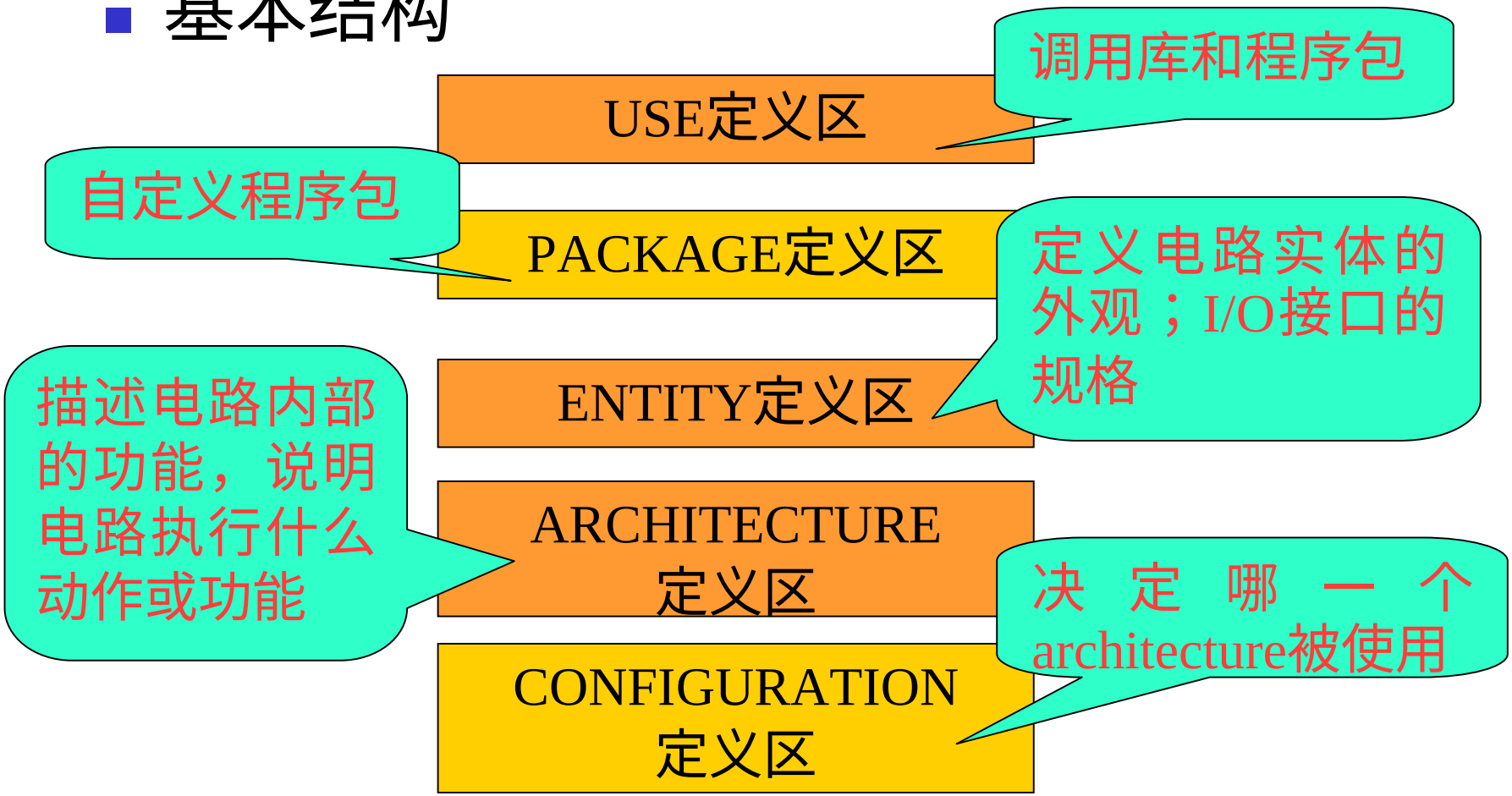


基本概念（2/7）

- VHDL的一些规定
  - 不区分大小写（‘’内的字符和“”内的字符串除外）；
  - 每个逻辑行以一个分号作为结束标志；
  - 注释说明使用双短划线“– –”开头（本行有效）；
  - 用户定义的变量名、实体名等必须以字母开头；
  - 下划线不能连用；
  - 命名不能与保留字相同；
  - 在MAXPLUSII中，存盘文件名应与设计的实体名相同。

基本概念（3/7）

- 基本结构

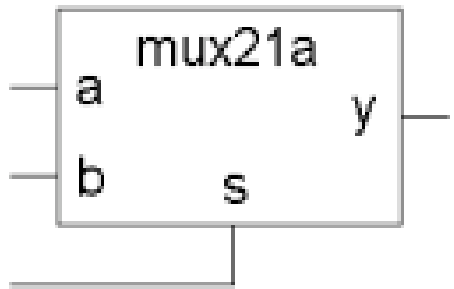

USE定义区
调用库和程序包
自定义程序包
PACKAGE定义区
定义电路实体的外观；I/O接口的规格
ENTITY定义区
描述电路内部的功能，说明电路执行什么动作或功能
ARCHITECTURE定义区
决定哪一个architecture被使用
CONFIGURATION定义区

基本概念（4/7）

- **2选1多路选择器的VHDL描述**

```
ENTITY mux21a IS
 PORT( a, b : IN  BIT ;
       s : IN  BIT;
       y : OUT BIT  ) ;
END ENTITY mux21a ;
```

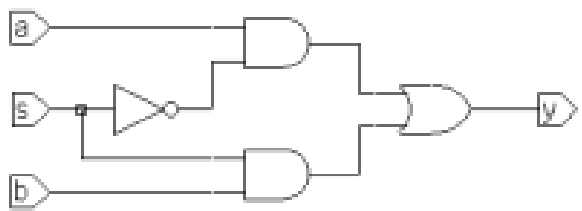
实体



mux21a实体

```
ARCHITECTURE one OF mux21a IS
 BEGIN
 y <=  a  WHEN  s = '0'  ELSE
         b  ;
END ARCHITECTURE one ;
```

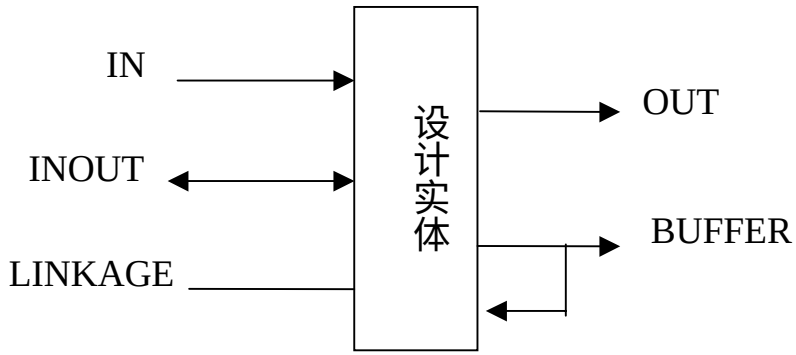
结构体



mux21a结构体

基本概念（5/7）

- 实体(entity)说明的相关概念
  - 类属参数说明
    - 在端口说明前面；
    - 例generic (delay: time);
  - 端口说明
    - 端口名，方向，数据类型
    - 方向包括： 输入(in)；输出(out)；双向(inout)；缓冲(buffer)；连接(linkage).



基本概念（6/7）

- 结构体(architecture)说明的相关概念
  - 结构体是由一个或多个并行语句构成的，他们的书写顺序并不代表他们的执行顺序。

```
Entity test1 Is
Port ( a, b : in bit;
       c, d : out bit);
end test1;
architecture test1_body of test1 is
begin
c <= a and b;
d <= a or b;
end test1_body;
```

输出只和输入有关

并行执行

```
Entity test1 Is
Port ( a, b : in bit;
       c, d : out bit);
end test1;
architecture test1_body of test1 is
begin
d <= a or b;
c <= a and b;
end test1_body;
```

与顺序无关

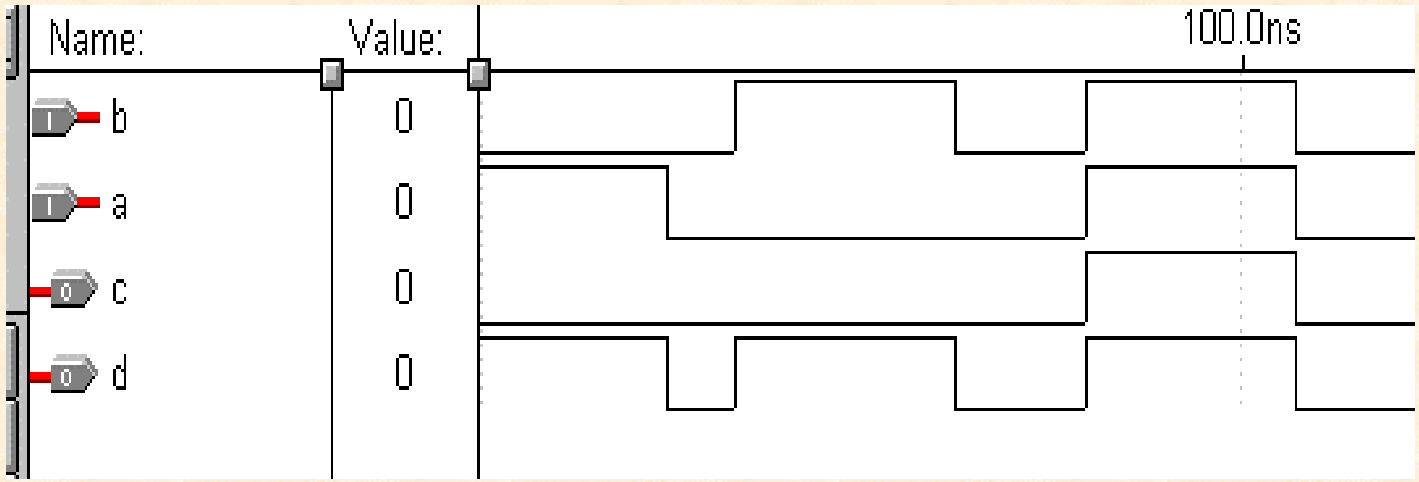
THE SAME

c <= a and b;
d <= a or b;

d <= a or b;
c <= a and b;

C = A and B

D = A OR B



基本概念（7/7）

- 所有的进程语句(process)都是并行执行的；
- 在一个进程语句中的代码是顺序执行的；
- 进程语句的输出与输入以及敏感信号表的事件发生有关；

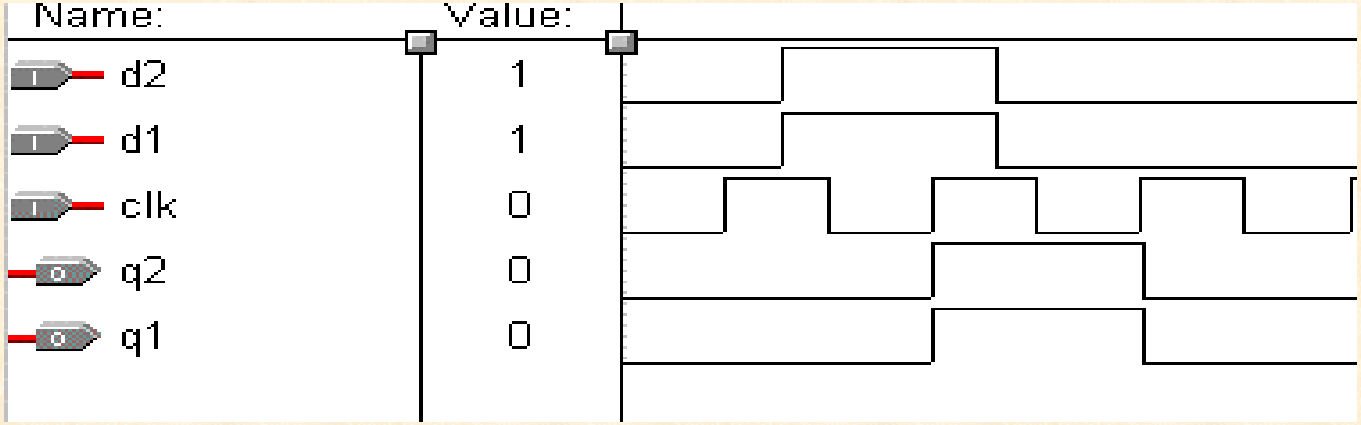
```
Entity test1 is
Port ( clk, d1, d2 : in bit;
       q1, q2 : out bit);
end test1;
architecture test1_body of
test1 is
begin
Process (clk, d1)
begin
if (clk'event and clk = '1')
then
q1 <= d1;
end if;
end process;
Process (clk, d2)
begin
if (clk'event and clk= '1') then
q2 <= d2;
end if;
end process;
end test1_body;
```

输出与有条件约束的输入有关

该代码在进程语句中顺序执行

```
Entity test1 is
Port ( clk, d1, d2 : in bit;
       q1, q2 : out bit);
end test1;
architecture test1_body of test1
is
begin
Process (clk, d2)
begin
if (clk'event and clk = '1')
then
q2 <= d2;
end if;
end process;
Process (clk, d1)
begin
if (clk'event and clk= '1') then
q1 <= d1;
end if;
end process;
end test1_body;
```

这两个进程语句并行执行




```
Process (clk, d1)
begin
if (clk'event and clk = '1') then
q1 <= d1;
end if;
end process;
```

```
Process (clk, d2)
begin
if (clk'event and clk = '1') then
q2 <= d2;
end if;
end process;
```

两个进程语句并行执行

VHDL的数据对象、类型及运算符

- VHDL语言的数据对象
  - 常量；变量；信号及文件；
- VHDL语言的数据类型
  - 标量类型；复合类型及子类型；
- VHDL语言的数据类型转换函数；
- VHDL语言的运算符
  - 逻辑、算术、关系及并置运算符；
- VHDL语言的词法单元
  - 注释、数字、字符及字符串、位串；

VHDL语言的数据对象（一）

- 常量
  - 指在设计实体中不会发生变化的值；
  - 可以是任何数据类型；
  - 可以在很多部分进行说明。
  - 说明的格式
    - Constant 常量名: 数据类型 [:=表达式]；
  - 例
    - Constant width_temp : integer :=8;
    - Constant VCC : real := 3.3;
    - Constant delay: time :=10ns;

VHDL语言的数据对象（二）

- 变量
  - 主要用于对暂时数据进行局部储存；
  - 是一个局部变量，只能在进程语句、过程语句、和函数语句的说明区域中加以说明；
  - 变量的赋值是直接的、立即生效的；
  - 要将一个变量的值用于作用范围之外时，需要将该变量的值赋给一个相同类型的信号；
  - 说明的格式
    - Variable 变量名: 数据类型 [:=表达式]；
    - 例: Variable temp : std_logic := ‘0’;

VHDL语言的数据对象（三）

- 信号
  - 是实体间动态交换数据的手段；
  - 在物理上它对应着硬件设计中的一条硬件连接线；
  - 初始值只对模拟有用，对综合来说意义不大；
  - 信号既可以作为输入/输出，也可以具有缓冲模式的特点。
  - 说明的格式
    - Signal 信号名：数据类型 [:=表达式]；
    - 例：signal clk：std_logic := ‘0’;

信号signal vs. 变量variable

信号**signal**	变量**variable**
代表电路中的连接	代表局部的一个暂存值
全局变量	局部变量
赋值符号‘**<=**’	赋值符号‘**:=**’
赋值在一个过程结束后才会变化	赋值没有延迟，立即变化

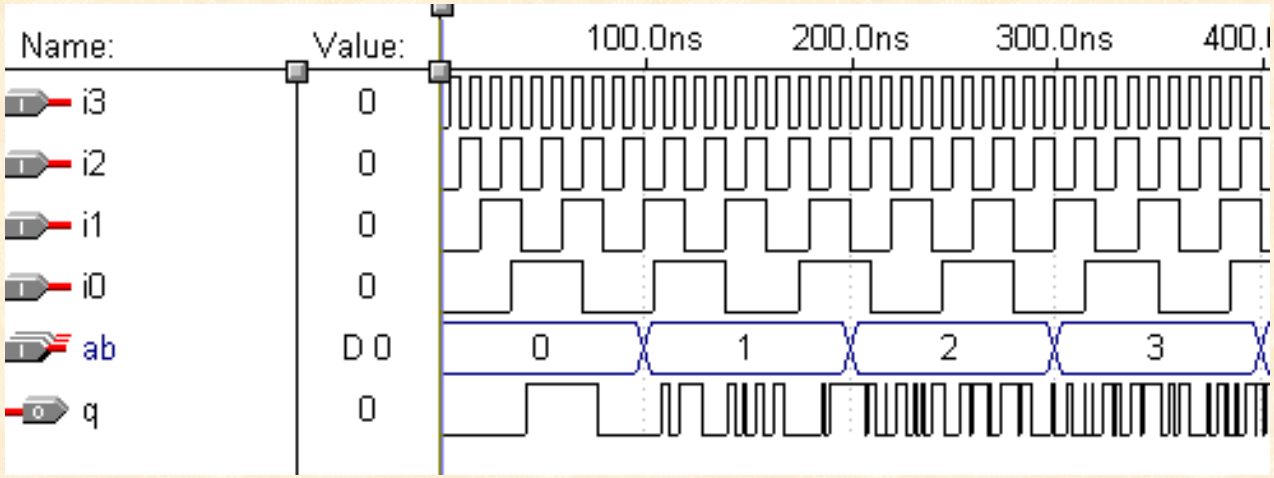
例：定义为信号

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

ENTITY mux4 IS
PORT (i0, i1, i2, i3, a, b : IN STD_LOGIC;
      q : OUT STD_LOGIC);
END mux4;

ARCHITECTURE body_mux4 OF mux4 IS
signal muxval : integer range 0 to 3;
BEGIN
process(i0,i1,i2,i3,a,b)
begin
muxval <= 0;
if (a = '1') then
   muxval <= muxval + 1;
end if;
if (b = '1') then
   muxval <= muxval + 2;
end if;
case muxval is
     when 0 => q <= i0;
     when 1 => q <= i1;
     when 2 => q <= i2;
     when 3 => q <= i3;
end case;
end process;
END body_mux4;
```

--注意信号定义的位置

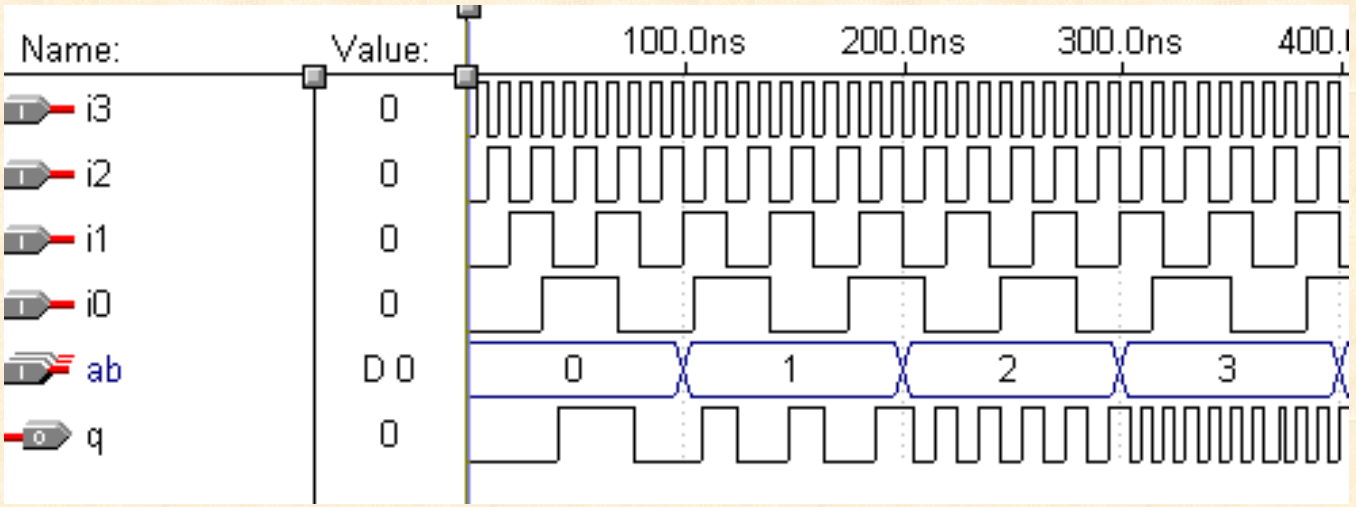


例：定义为变量

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

ENTITY mux4 IS
PORT (i0, i1, i2, i3, a, b : IN STD_LOGIC;
      q : OUT STD_LOGIC);
END mux4;

ARCHITECTURE body_mux4 OF mux4 IS
BEGIN
process(i0,i1,i2,i3,a,b)
variable muxval : integer range 0 to 3;  --注意变量定义的位置
begin
muxval := 0;
if (a = '1') then
   muxval := muxval + 1;
end if;
if (b = '1') then
   muxval := muxval + 2;
end if;
case muxval is
     when 0 => q <= i0;
     when 1 => q <= i1;
     when 2 => q <= i2;
     when 3 => q <= i3;
end case;
end process;
END body_mux4;
```



VHDL语言的数据对象（四）

- 文件
  - VHDL-93语法把文件也当作对象；
  - 不能被赋值；
  - 通过规定的过程和函数对文件对象进行读出和写作的操作；
  - 说明的格式
    - File 文件名：文件类型 is [方向] 路径表达式；
    - Type 文件类型名 is file of 数据类型；
    - 例: type filetype is file of std_logic_vector;
      file myfile : filetype is in “/myproject/vector.in”

VHDL语言的数据类型（1/12）

- VHDL是一种强类型语言
  - 每个数据对象只能具有一个数据类型，且只能具有那个数据类型的值；
  - 对某数据对象进行操作的类型必须与该对象的类型相匹配；
  - 不同类型之间的数据不能直接代入；
  - 相同数据类型时，位长不同的也不能代入；
  - 文件类型与存取类型主要用于建立模拟模型。

VHDL语言的数据类型（2/12）

- 标量类型
  - 主要用来描述一次持有一个值的对象的基本数据类型：
    - 整数、浮点数、可枚举和物理类型
  - 整数类型：integer
    - 范围-（2^{31}-1）到+（2^{31}-1）；
    - 不能使用逻辑操作符；
    - 例： constant max : integer :=128;
      signal number : integer range 0 to 255;
      variable int8 : integer;
      type digital is integer range 0 to 9;

VHDL语言的数据类型（3/12）

- 浮点数类型：real
  - 可表示大部分实数，包括整数值和分数值；
  - 常用来进行算法研究的描述；
  - 范围：-1.0E38 到 +1.0E38;
  - 例： constant max：real：=128.0;
    signal number：real range 0.0 to 255.0;
    variable voltage：real;
    type current is real range –1.0e3 to +1.0e3;

VHDL语言的数据类型（4/12）

- 可枚举类型
  - 一组表列形式给定的适用于特定操作所需要的值；
  - 定义格式：
    - type 枚举数据类型名 is (枚举元素，枚举元素，……);
  - 元素的大小：左边的值小于右边的值；
  - 可应用于微处理器指令集以及状态机的状态表；
  - 例： type boolean is (false, true);
    type bit is (‘0’, ‘1’);

VHDL语言的数据类型（5/12）

- 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
  );
- Std_logic是std_ulogic的子类型

VHDL语言的数据类型（6/12）

- 物理类型
  - 用来表示象时间、电压等这样的物理量；
  - 定义格式：type 物理数据类型名 is 约束范围
    units
    基本单位；
    单位条目；
    end units;
  - 物理类型说明的约束范围规定了按基本单位能表示的物理类型的最小值和最大值；
  - 所有的单位标识必须是唯一的。

VHDL语言的数据类型（7/12）

- 在VHDL中，唯一的预定义的物理类型是时间：

```
Type time is range 0.0 to +1.0e19
Units
    fs;
    ps=1000fs;
    ns=1000ps;
    us=1000ns;
    ms=1000us;
    sec=1000ms;
    min=60sec;
    hr=60min;
End units;
```

VHDL语言的数据类型（8/12）

- 复合类型
  - 可以在某个时刻持有多个值；
  - 由数组类型和记录类型组成，它们中的元素还是标量类型的元素：
    - 数组类型是由相同类型的多个标量元素组成，即同构复合类型；
    - 记录类型是由不同类型的标量元素组成的集合，即异构复合类型。

VHDL语言的数据类型（9/12）

- 数组类型
  - 可以含有一个下标的一维数组，也可以是含有多个下标的多维数组;
  - 二维数组常用于建立真值表；
  - 定义的格式：
    - type 数组类型名 is array 约束范围 of 数组元素类型
  - 根据约束范围的不同，可以是限定性的也可以是非限定性的；

例：type word is array (15 downto 0) of bit;

type std_logic_vector is array (natural range <>) of logic;

VHDL语言的数据类型（10/12）

- 记录类型
  - 记录元素可以是任何类型的标量元素，可以属于相同的也可以是不同的类型。
  - 定义的格式：
    - Type 记录类型名 is record
      记录元素名1 ：数据类型名；
      记录元素名2 ：数据类型名；
      ……
      end record;

VHDL语言的数据类型（11/12）

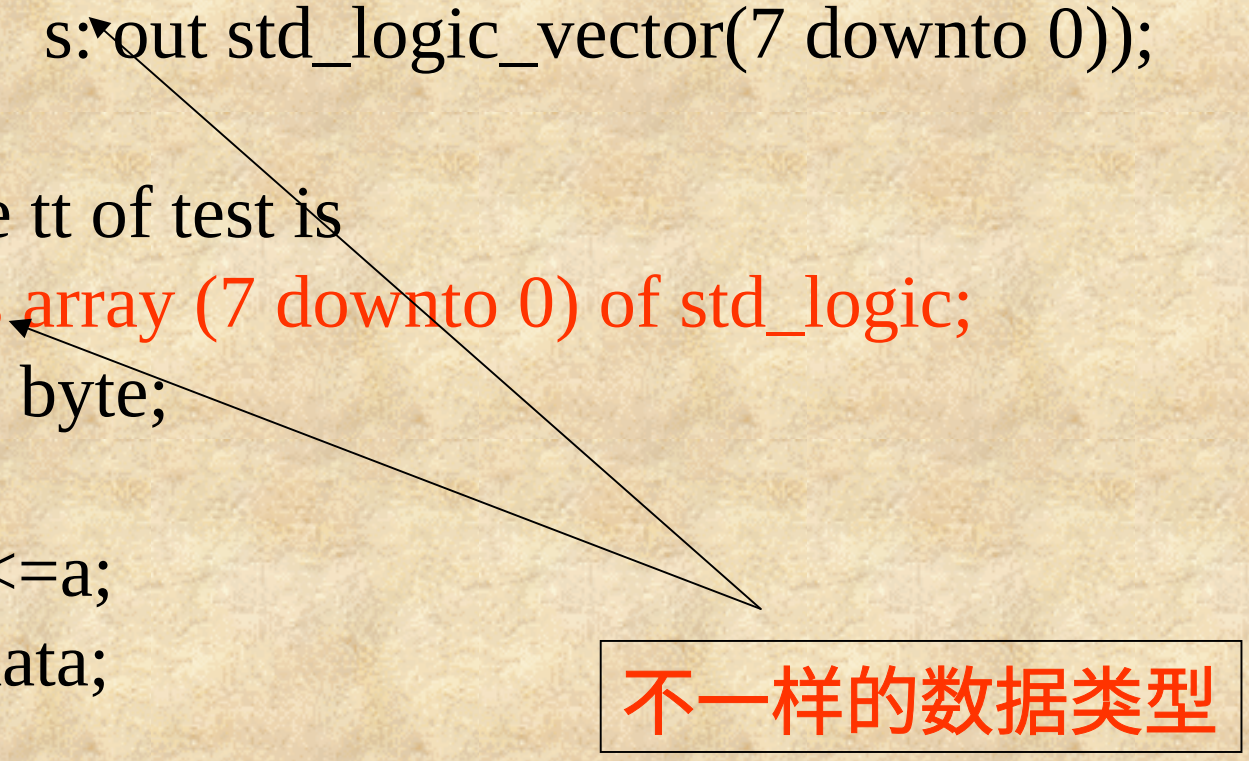
- 子类型
  - 上述基本类型的子集；
  - 对选择信号赋值语句或case语句进行约束；
  - 建立决断子类型；
  - 子类型与其基本类型之间可以相互赋值；
    - 如果基本类型向子类型赋值，该基本类型对象的值要在子类型的范围内。
  - 不能用array来定义新的子类型；
  - 定义的格式
    - **Subtype** 子类型名 **is** 基本数据类型名 **[**范围限制**]**

VHDL语言的数据类型（12/12）

- 正确的例子：
  - Subtype natural is integer range 0 to +21474836;
  - Subtype byte is std_logic_vector(7 downto 0);
  - Subtype std_logic is resolved std_ulogic;
  - Subtype X01 is resolved std_ulogic range ‘X’ to ‘1’;
- 错误的例子：
  - Subtype byte is array (7 downto 0) of std_logic;
- **TYPE与SUBTYPE的区别**
  - TYPE定义的类型是一种全新的类型；
  - SUBTYPE仅仅是某基本类型的子集；

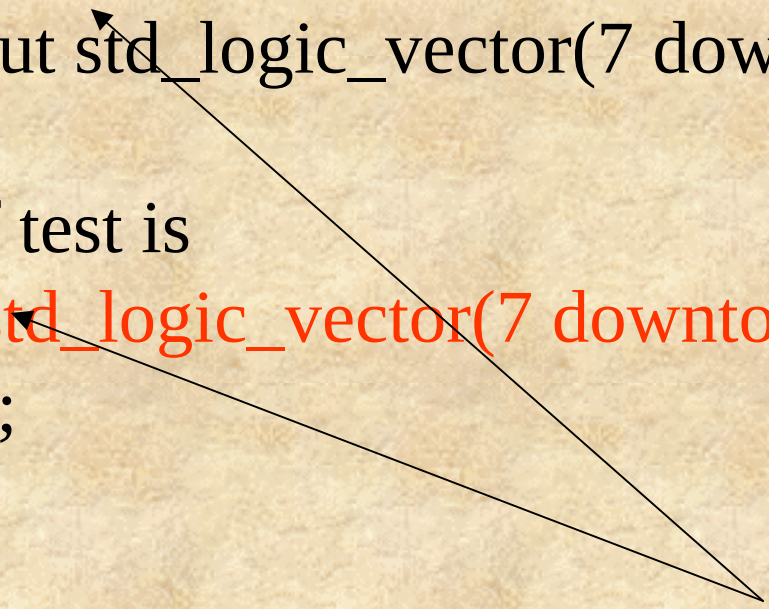
一个错误的例子

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
        port (a: in std_logic_vector(7 downto 0);
                s: out std_logic_vector(7 downto 0));
end test;
architecture tt of test is
type byte is array (7 downto 0) of std_logic;
signal data: byte;
begin
        data<=a;
        s<=data;
end tt;
```



修改后

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
        port (a: in std_logic_vector(7 downto 0);
                s: out std_logic_vector(7 downto 0));
end test;
architecture tt of test is
subtype byte is std_logic_vector(7 downto 0);
signal data: byte;
begin
        data<=a;
        s<=data;
end tt;
```



相同的数据类型

VHDL的预定义数据类型

1. 布尔(BOOLEAN)数据类型
2. 位(BIT)数据类型
3. 位矢量(BIT_VECTOR)数据类型
4. 字符(CHARACTER)数据类型
5. 整数(INTEGER)数据类型
6. 实数(REAL)数据类型

VHDL的预定义数据类型

7. 字符串(STRING)数据类型

```
VARIABLE string_var : STRING (1 TO 7 ) ;
string_var := "a b c d" ;
```

8. 时间(TIME)数据类型

```
TYPE time IS RANGE －2147483647 TO 2147483647
    units
        fs ;                        -- 飞秒，VHDL中的最小时间单位
        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 = 60 min ;               -- 时
end units ;
```

IEEE预定义标准逻辑位与矢量

1．标准逻辑位STD_LOGIC数据类型

2．标准逻辑矢量(STD_LOGIC_VECTOR)
数据类型

STD_LOGIC_VECTOR类型定义如下：

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

其他预定义标准数据类型

1. 无符号数据类型(UNSIGNED TYPE)

十进制的8可以作如下表示：

```
UNSIGNED'("1000")
```

两则无符号数据定义的示例：

```
VARIABLE var : UNSIGNED(0 TO 10) ;
SIGNAL   sig : UNSIGNED(5 TO 0) ;
```

2. 有符号数据类型(SIGNED TYPE)

例如：

SIGNED'("0101") 代表 +5，5

SIGNED'("1011") 代表 –5

数据类型转换函数（一）

- CONV_INTEGER(参数)；
  - 将其他类型的数据转换成整数类型
    - INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC；
  - 操作数的转换范围是 -2147483647 to 2147483647,
    - UNSIGNED数据是31-bit；
    - SIGNED数据是32-bit.
- CONV_UNSIGNED(参数，位数);
  - 将其他类型的数据转换成一定位数的无符号类型
    - INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC；

数据类型转换函数（二）

- CONV_SIGNED(参数，位数)
  - 将其他类型的数据转换成一定位数的有符号类型
    - INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC；
- CONV_STD_LOGIC_VECTOR(参数，位数)
  - 将其他类型的数据转换成一定位数的标准逻辑矢量类型；
    - INTEGER, UNSIGNED, SIGNED, or STD_LOGIC；

数据类型转换的例子

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY adder IS
        PORT (op1, op2 : IN UNSIGNED(7 DOWNTO 0);
                result   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
                        );
END adder;
ARCHITECTURE maxpld OF adder IS
BEGIN
        result <= CONV_STD_LOGIC_VECTOR(op1 + op2, 8);
END maxpld;
```

VHDL语言的运算操作符（1/5）

- 主要有四种运算操作符
  - 逻辑运算符；
  - 算术运算符；
  - 关系运算符；
  - 并置运算符；
- 操作数的类型应该和运算操作符所要求的**类型一致**。

VHDL语言的运算操作符（2/5）

类　型	操作符	功　能	操作数数据类型
算术操作符	+	加	整数
	—	减	整数
	&	并置	一维数组
	*	乘	整数和实数(包括浮点数)
	/	除	整数和实数(包括浮点数)
	MOD	取模	整数
	REM	取余	整数
	SLL	逻辑左移	BIT 或布尔型一维数组
	SRL	逻辑右移	BIT 或布尔型一维数组
	SLA	算术左移	BIT 或布尔型一维数组
	SRA	算术右移	BIT 或布尔型一维数组
	ROL	逻辑循环左移	BIT 或布尔型一维数组
	ROR	逻辑循环右移	BIT 或布尔型一维数组
	**	乘方	整数
	ABS	取绝对值	整数

接上页

关系操作符	=	等于	任何数据类型
	/=	不等于	任何数据类型
	<	小于	枚举与整数类型，及对应的一维数组
	>	大于	枚举与整数类型，及对应的一维数组
	<=	小于等于	枚举与整数类型，及对应的一维数组
	>=	大于等于	枚举与整数类型，及对应的一维数组
逻辑操作符	AND	与	BIT，BOOLEAN，STD_LOGIC
	OR	或	BIT，BOOLEAN，STD_LOGIC
	NAND	与非	BIT，BOOLEAN，STD_LOGIC
	NOR	或非	BIT，BOOLEAN，STD_LOGIC
	XOR	异或	BIT，BOOLEAN，STD_LOGIC
	XNOR	异或非	BIT，BOOLEAN，STD_LOGIC
	NOT	非	BIT，BOOLEAN，STD_LOGIC
符号操作符	+	正	整数
	—	负	整数

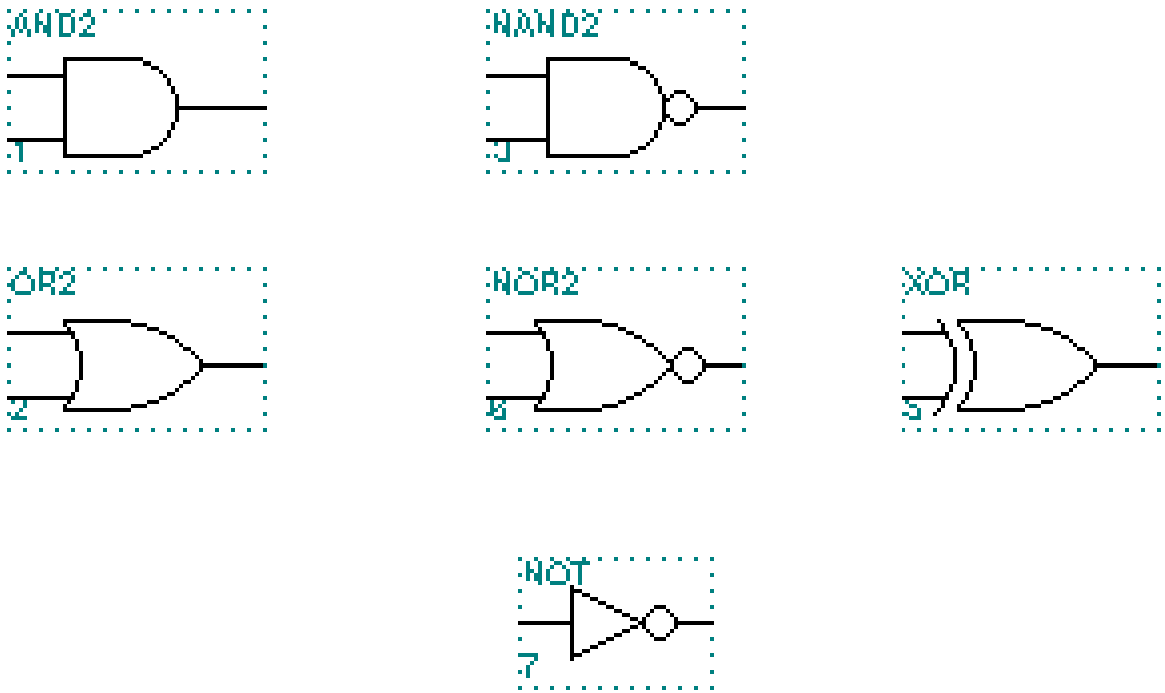
VHDL语言的运算操作符（3/5）

运算符	优先级
NOT, ABS, **	最高优先级
*, /, MOD, REM	
+(正号), -(负号)	
+, -, &	
SLL, SLA, SRL, SRA, ROL, ROR	
=, /=, <, <=, >, >=	
AND, OR, NAND, NOR, XOR, XNOR	最低优先级

VHDL操作符优先级

VHDL语言的运算操作符（4/5）

- AND
- NAND
- OR
- NOR
- XOR
- NOT



逻辑运算操作符

例

```
 SIGNAL a, b, c: STD_LOGIC_VECTOR (3 DOWNTO 0) ;
 SIGNAL d, e, f, g: STD_LOGIC_VECTOR(1 DOWNTO 0);
 SIGNAL h, I, j, k : STD_LOGIC ;
 SIGNAL l, m, n, o, p : BOOLEAN ;
 ...
 a<=b AND c;
--b、c 相与后向a赋值，a、b、c的数据类型同属4位长的位矢量

 d<=e OR f OR g ;
-- 两个操作符OR相同，不需括号

 h<=(i NAND j)NAND k ;
-- NAND不属上述两种算符中的一种，必须加括号

l<=(m XOR n)AND(o XOR p);
-- 操作符不同，必须加括号
```

```
SIGNAL a, b, c: STD_LOGIC_VECTOR (3 DOWNTO 0) ;
SIGNAL d, e, f, g: STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL h, I, j, k : STD_LOGIC ;
SIGNAL l, m, n, o, p : BOOLEAN ;
```

```
h<=i AND j AND k ;
-- 两个操作符都是AND，不必加括号

h<=i AND j OR k ;
-- 两个操作符不同，未加括号，表达错误

 a<=b AND e ;
-- 操作数b 与 e的位矢长度不一致，表达错误

 h<=i OR l ;
-- i 的数据类型是位STD_LOGIC，而l的数据类型是
-- 布尔量BOOLEAN，因而不能相互作用，表达错误。
```

VHDL语言的运算操作符（5/5）

- 并置运算符: **&**
  - 主要用于位和位矢量的连接；
    - 就是将并置运算符右边的内容连接在左边的内容之后，形成一个新的数组；
  - 可以将两个位连接起来形成一个位矢量；
  - 可以将两个位矢量连接起来形成一个新的位矢量；
  - 可以将位矢量和位连接起来。

位的连接的几种方法

```
Signal a, b, c, d : std_logic;
Signal q : std_logic_vector(3 downto 0);

q<=a&b&c&d;                    --正确使用并置运算符；

q<=(a, b, c, d);               --另一种位连接的方法；

q<=(3=>a, 2=>b, 1=>c, 0=>d);
        --指定位的脚标来进行位连接；

q<=a&a&c&d;        --如果相同的信号要使用多次；

q<=(1=>c, 0=>d, others=>a);
        --采用others进行位的连接；注意others的说
        --明只能放在最后。
```

VHDL的词法单元（1/3）

- 标识符
  - 有效的字符：包括26个大小写英文字母，数字包括0～9 以及下划线“_”；
  - 任何标识符必须以英文字母开头；
  - 必须是单一下划线“_”，且其前后都必须有英文字母或数字；
  - 标识符中的英语字母不分大小写；
  - 允许包含图形符号(如回车符、换行符等)，也允许包含空格符；

VHDL的词法单元（2/3）

- 注释
  - 以虚线‘--’开始直到本行末尾的一段文字；
  - 主要用来对设计进行说明和解释；
  - 可以在源代码的任何位置进行标注；
  - 注释文字不会被编译；
- 数字
  - 可以表示成十进制、二进制、八进制或十六进制；
  - 可以是整数，也可以是浮点数；
  - 相邻数字之间插入一个下划线，或在数字前加若干个零，对数字的数值不会有影响；
  - 但是不允许在数字之间出现空格。

数 字

整数：156E2(=15600)，　　45_234_287 (=45234287)；

实数：必须带有小数点，如：1.335，1.0;

44.99E-2(=0.4499)；

88_670_551.453_909(=88670551.453909);

以数制基数表示的文字：基#基于基的整数#[指数]

```
SIGNAL d1,d2,d3,d4,d5, : INTEGER RANGE 0 TO
255;
d1<=110#170# ;          -- (十进制表示，等于 170)
d2<=16#FE#    ;         -- (十六进制表示，等于 254)
d3<=2#1111_1110#;       -- (二进制表示，等于 254)
d4<=8#376#;             -- (八进制表示，等于 254)
d5<=16#E#E1; -- (十六进制表示，等于2#1110000#，等于224)
```

VHDL的词法单元（3/3）

- 字符和字符串
  - 字符是用单引号括起来的ASCII码；
  - 字符串是用双引号括起来的字符序列；
    - ‘A’和‘a’是不相同的；大小写不同；
    - “a”和‘a’是不相同的；双引号是字符串，单引号是字符；
- 数字位串
  - 用双引号括起来的数字序列，序列前有一个基数说明符；
  - 二进制时，可以省略基数说明符‘B’；

字符和字符串

“it is time out“，’x’， "X“，"BB$CC"

数位字符串

B：二进制基数符号，表示二进制位0或1，在字符串中的每位表示一个Bit, 可以省略。

O：八进制基数符号，在字符串中的每一个数代表一个八进制数，即代表一个3位(BIT)的二进制数。

X：十六进制基数符号(0～F)，代表一个十六进制数，即一个4位的二进制数。

```
d1<=B"1_1101_1110“   --二进制数数组，位矢数组长度是9
d2<=O"15"            --  八进制数数组，位矢数组长度是6
d3<=X"AD0"           --  十六进制数数组，位矢数组长度是12
d4<=“101_010_101_010”    --系统默认为是二进制数数组
d5 <= "0AD0"             --表达错误，缺X。
```

VHDL的句法

- 顺序描述语句
  - 顺序语句只能出现在进程、过程和函数中；
  - 定义在进程、过程和函数中所执行的算法；
  - 执行顺序完全按照源代码中语句的编写顺序来执行；
  - 用于描述时序逻辑电路；
- 并行描述语句
  - 执行顺序与源代码中语句的编写顺序无关；
  - 用于描述组合逻辑电路；

VHDL中的顺序语句（1/10）

- 信号代入与变量赋值
  - 信号代入：目标信号<=表达式;
    - 在进程、函数、过程中使用该语句时，它是一条顺序语句；
  - 变量赋值：目标变量:=表达式;
- Wait 语句
  - 使用wait语句时，process后面不再有敏感信息表；
  - Wait on语句
    - 格式：Wait on 敏感信号，[敏感信号……]；
    - 功能：使进程挂起，直到某个敏感信号变化；

VHDL中的顺序语句（2/10）

- Wait until语句
  - 格式：Wait until 条件表达式；
  - 功能：将进程挂起，直到表达式返回一个“true”;
  - 该表达式建立了一个隐含的敏感信号表；
- Wait for语句
  - 格式：Wait for 时间表达式；
  - 功能：进程执行到该语句时挂起，直到时间表达式规定的时间到时，进程继续执行后面的语句；
  - Wait for语句只用于模拟，不能被综合。

VHDL中的顺序语句（3/10）

- If 语句
  - 格式：if <条件> then
    <顺序语句>;
    [elsif <条件> then
    <顺序语句>;]
    ……
    [else
    <顺序语句>;]
    end if

可以有多个elsif子句

只可以有一个else子句

注意这里有空格

VHDL中的顺序语句（4/10）

- Case 语句
  - 可读性比if语句好；常用来描述根据某逻辑表达式的值而进行的操作；
  - 格式：case <表达式> is
    when <选择> => <顺序语句>；
    ……
    when others => <顺序语句>;
    end case:
  - When语句中不能有相同的选择；
  - When语句中的选择值必须被穷举，不能遗漏；
  - When语句可以颠倒次序，但是others必须在最后。

VHDL中的顺序语句（5/10）

- Loop语句
  - 用于某些操作重复进行或重复进行到某个条件满足为止的情况。
    - For loop循环；while loop循环；
  - For loop循环
    - 格式：[循环标号:] for 循环变量 in 离散范围 loop
      <顺序处理语句>;
      end loop [循环标号]；
    - 离散范围值可以不是整数，只要是离散数值即可。

VHDL中的顺序语句（6/10）

- While loop循环
  - 格式：[循环标号:] while <条件> loop
    <顺序处理语句>;
    end loop [循环标号]；
- for loop语句一般可以被综合；while loop语句只有某些高级综合工具可以综合；
- 一般采用for loop语句。

VHDL中的顺序语句（7/10）

- Next语句
  - 格式：next [循环标号] [when 条件]；
  - 功能：有条件或无条件地结束当前这次循环，开始下一次循环；
  - 用在loop循环内部；
- Exit语句
  - 格式： exit [循环标号] [when 条件]；
  - 功能：有条件或无条件地结束当前这次循环，并终止该循环，执行loop循环语句后面的语句；
  - 用在loop循环内部；

VHDL中的顺序语句（8/10）

- Return语句
  - 格式：return [表达式]；
  - 用来结束当前的函数或是过程体的执行；
  - 在过程体中，该语句不能有表达式；
  - 在函数体中，该语句必须有表达式；
- Null语句
  - 格式：null;
  - 空操作，常用于case语句的when others=>null;

VHDL中的顺序语句（9/10）

- 过程调用语句
  - 格式：过程名 （实参表）；
  - 在进程、函数、过程中使用该语句时，它是一条顺序语句；
- 断言语句
  - 格式：assert <条件>
    report <输出信息>
    severity <出错级别>;

注意只有最后有一个分号

VHDL中的顺序语句（10/10）

- 断言语句的出错级别：
  - {note, warning, error, failure};
- 该语句可以中断模拟过程，提供相应的信息；

- Report语句
  - 格式：report <输出信息>
    [severity <出错级别>];
  - VHDL-93提供的一种顺序断言语句的短格式。

并行描述语句

并行描述语句（1/12）

- 进程语句: process
  - 格式： [进程标号:] process [敏感信号表] [is]
    [进程语句说明部分]；
    begin
    <进程描述部分>
    end process [进程标号]；
  - 所有的进程语句都是并行执行的；
  - 在一个进程语句中的代码是顺序执行的；
  - 进程语句的输出与输入以及敏感信号表的事件发生有关；
    - 任何进程描述中最好有一个敏感信号表；

并行描述语句（2/12）

- 块语句 ：block
  - 格式：[块标号:]block [卫式表达式]
    [类属句子[类属接口表;]];
    [端口句子[端口接口表;]];
    [块说明部分]；
    begin
    <块描述部分>;
    end block [块标号]；
  - 每个块语句都是结构体的子模块；
  - 在一个块语句中的代码也是并行执行的；
  - 卫式表达式为假时，不执行该块语句；
    - 含卫式表达式的块语句不能被综合；

并行描述语句（3/12）

- 信号代入语句
  - 简单的信号赋值：<目标信号> <=<表达式>；
  - 有条件的信号赋值:
    - 格式：
      <目标信号> <= <表达式1> when <条件1> else
      <表达式2> when <条件2> else
      ……
      <表达式n-1> when <条件> else
      <表达式n>;

If...then语句 vs. When语句

If 语句是顺序语句	When语句是并行语句
只能在进程中使用	必须用在进程之外
Else语句可有可无	Else语句是必须有的
可以嵌套	不能嵌套
高级描述，与硬件无关	与硬件电路十分贴近

并行描述语句（4/12）

- 选择性的信号赋值
  - 格式：
    with <表达式> select
    <目标信号> <= <表达式1> when <条件1>，
    <表达式2> when <条件2>，
    ……
    <表达式n> when <条件n>，
    [<表达式n+1> when others];
  - 与case语句类似，但是是并行描述语句；
  - 必须把表达式的值在条件中都要列举出来；

并行描述语句（5/12）

- 并行断言语句
  - 格式与顺序断言语句相同；
  - 在断言条件为“false”时，给出一条信息报告；
- 并行过程调用语句
  - 格式与顺序过程调用语句相同；
  - 并行执行，执行顺序与书写顺序无关；

并行描述语句（6/12）

- 元件例化语句
  - 主要用于模块化的设计当中，避免大量的重复工作；
  - 引用元件时，首先在结构体说明部分进行元件的说明（component）；
  - 然后在使用元件时，对该元件进行例化描述；
  - 各个例化语句的执行顺序与书写顺序无关；

并行描述语句（7/12）

- 元件说明部分
  - 第一部分是将一个现成的设计实体定义为一个元件；
  - 语句的功能是对待调用的元件作出调用声明；
  - 格式：**COMPONENT <元件名>**
    [generic <参数说明>;]
    PORT (端口名表) ；
    END COMPONENT；

并行描述语句（8/12）

- 元件例化部分
  - 格式：
    <标号名> : <元件名> [GENERIC MAP(参数映射)]
    PORT MAP([端口名 =>] 连接端口名,...);
  - 此元件与当前设计实体(顶层文件)中元件间及端口的连接说明;
  - 标号名在结构体中必须是唯一的；
  - 映射语句就是把元件的参数和端口与实际连接的信号对应起来，以进行元件的引用；
  - 映射方式：位置映射和名称映射；

并行描述语句（9/12）

- GENERIC类属参数说明及映射语句
  - GENERIC类属参数说明语句
    - 格式: GENERIC(常数名:数据类型[:=设定值]
      [;常数名:数据类型[:=设定值].....]);
    - 用在实体说明或元件说明中，端口说明语句之前；
    - 主要用来为设计实体指定参数；
      - 端口宽度、器件延迟时间等；

并行描述语句（10/12）

- GENERIC类属参数映射语句
  - 格式： **GENERIC MAP**（类属表）
  - 类属映射语句可用于设计从外部端口改变元件内部参数或结构规模的元件，或称类属元件；
  - 易于使设计具有通用性；
  - 类属映射语句与端口映射语句PORT MAP()具有相似的功能和使用方法；
  - 它描述相应元件类属参数间的衔接和传送方式，它的类属参数衔接（连接）表达方式也相同。

```
LIBRARY IEEE;
  USE IEEE.STD_LOGIC_1164.ALL;
  ENTITY andn IS
    GENERIC(n: INTEGER:=2; delay: time);
      --定义类属参量及其数据类型
    PORT(a : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
      --用类属参量限制矢量长度
           c : OUT STD_LOGIC);
   END;
   ARCHITECTURE behave OF andn IS
      BEGIN
        PROCESS (a)
            VARIABLE int : STD_LOGIC;
        BEGIN
           int := '1';
              FOR I IN a'LENGTH - 1 DOWNTO 0 LOOP
                IF a(i)='0' THEN   int := '0';
                END IF;
              END LOOP;
                c <=int after (delay);--延迟赋值语句
        END PROCESS;
   END;
```

底层文件

无需赋初值

需要赋初值

顶层文件

```
LIBRARY IEEE;
    USE IEEE.STD_LOGIC_1164.ALL;
    ENTITY exn1 IS
       PORT(d1, d2, d3 : IN STD_LOGIC;
            d4: IN STD_LOGIC_vector(4 downto 0);
            q1, q2  : OUT STD_LOGIC);
    END;
    ARCHITECTURE exn_behav OF exn1 IS
     COMPONENT andn                          --元件调用声明
          GENERIC ( n : INTEGER; delay : time);
          PORT(a: IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
               c: OUT STD_LOGIC);
     END COMPONENT ;
    BEGIN
      u1: andn GENERIC MAP (n=>3, delay =>10 ns)
      -- 类属映射语句，定义类属变量，n赋值为2,延迟为10 ns;
          PORT MAP (a(0)=>d1,a(1)=>d2,a(2)=>d3, c=>q1);
      u2: andn GENERIC MAP (5, 20 ns) --n赋值为5,延迟为20ns
          PORT MAP (d4,q2);
END;
```

名称映射

必须有空格

位置映射

并行描述语句（11/12）

- 生成语句（generate）
  - 可以建立重复结构或者是在多个模块的表示形式之间进行选择的语句；
  - For 模式生成语句
    - 用来进行重复结构的描述；
    - 格式：

标号：**FOR** 循环变量 **IN** 取值范围 **GENERATE**
BEGIN
<并行描述语句>；
END GENERATE [标号] ；

并行描述语句（12/12）

- If 模式生成语句
  - 用来描述一个结构中的例外情况；
  - 格式：

```
标号： IF <条件> GENERATE
        Begin
        <并行描述语句>;
       END GENERATE [标号] ；
```

For loop语句 vs. For generate语句

For loop语句	For generate语句
顺序描述语句	并行描述语句
允许next,exit控制循环	不允许使用next,exit 语句

If…then语句 vs. If…generate语句

If…then语句	If…generate语句
顺序描述语句	并行描述语句
允许使用else语句	不允许使用else语句
无标号	有标号

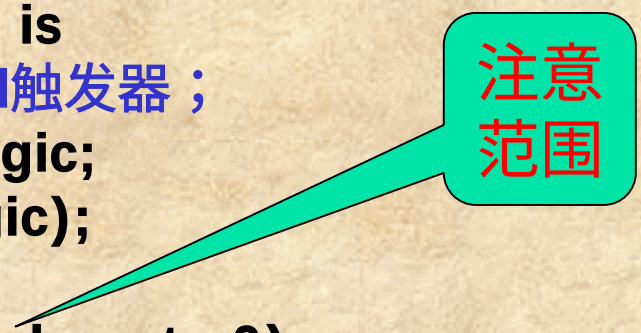
四位移位寄存器的VHDL描述—普通描述

```
library ieee;
use ieee.std_logic_1164.all;
entity shift_reg1 is
        port (d1,cp: in std_logic;
                d0: out std_logic);
end shift_reg1;
architecture structure of shift_reg1 is
        component dff          --利用d触发器；
                port (d, clk: in std_logic;
                        q: out std_logic);
        end component;
        signal q: std_logic_vector(2 downto 0);
Begin                                   --描述部分；
        dff1: dff port map (d1, cp, q(0));
        dff2: dff port map (q(0), cp, q(1));
        dff3: dff port map (q(1), cp, q(2));
        dff4: dff port map (q(2), cp, d0);
end structure;
```

注意范围

四位移位寄存器的VHDL描述—一致化描述

```
library ieee;
use ieee.std_logic_1164.all;
entity shift_reg2 is
        port (d1,cp: in std_logic;
                d0: out std_logic);
end shift_reg2;
architecture structure of shift_reg2 is
        component dff          --利用d触发器；
                port (d, clk: in std_logic;
                        q: out std_logic);
        end component;
        signal q: std_logic_vector(4 downto 0);
Begin                                   --描述部分；
        q(0)<=d1;
        dff1: dff port map (q(0), cp, q(1));
        dff2: dff port map (q(1), cp, q(2));
        dff3: dff port map (q(2), cp, q(3));
        dff4: dff port map (q(3), cp, q(4));
        d0<=q(4);
end structure;
```



四位移位寄存器的VHDL描述—for生成语句描述

```
library ieee;
use ieee.std_logic_1164.all;
entity shift_reg3 is
        port (d1,cp: in std_logic;
                d0: out std_logic);
end shift_reg3;
architecture structure of shift_reg3 is
        component dff          --利用d触发器；
                port (d, clk: in std_logic;
                        q: out std_logic);
        end component;
        signal q: std_logic_vector(4 downto 0);
Begin                                   --描述部分；
        q(0)<=d1;
        label1: for i in 0 to 3 generate
                dffx: dff port map (q(i), cp, q(i+1));
        end generate label1;
        d0<=q(4);
end structure;
```

四位移位寄存器的VHDL描述—含有if生成语句的描述

```
library ieee;
use ieee.std_logic_1164.all;
entity shift_reg1 is
        port (d1,cp: in std_logic; d0: out std_logic);
end shift_reg1;
architecture structure of shift_reg1 is
        component dff  --利用d触发器；
          port (d, clk: in std_logic; q: out std_logic);
        end component;
        signal q: std_logic_vector(3 downto 1);
Begin                                          --描述部分；
        label1: for i in 0 to 3 generate
                l1: if (i=0) generate
                        dffx: dff port map (d1, cp, q(i+1));
                end generate;
                l2: if (i=3) generate
                        dffx: dff port map (q(i), cp, d0);
                end generate;
                l3: if (i/=0) and (i/=3) generate
                        dffx: dff port map (q(i), cp, q(i+1));
                end generate;
        end generate label1;
end structure;
```

需要标号

子程序结构(1/5)

- 子程序就是在主程序调用它以后能够将处理结果返回主程序的程序模块。
- 可以被反复调用；
- VHDL中的子程序
  - 过程(procedure)
  - 函数(function)

子程序结构（2/5）

- 过程（PROCEDURE）
  - 格式：
    - 过程说明： PROCEDURE <过程名>
      ([对象种类] 参数名[,参数名…]:[方式] 数据类型;
      [对象种类] 参数名[,参数名…]:[方式] 数据类型;
      ……);
    - 过程定义：PROCEDURE <过程名>(参数表) IS
      [说明部分]；
      BIGIN
      <顺序语句>;
      END [PROCEDURE] <过程名>;

子程序结构（3/5）

- 参数种类：变量、常量和信号；
- 参数方式：
  - 输入in：默认参数种类为常量；
  - 输出out和inout：默认参数种类为变量；
- 说明部分：
  - 包括变量说明、常量说明和类型说明；
  - 过程中不能定义新的信号；
- 例：

```
PROCEDURE  pro1 (VARIABLE  a, b :  INOUT REAL) ;
PROCEDURE  pro2 (CONSTANT  a1 :  IN INTEGER  ;
                 SINGAL  b1 :  OUT BIT ) ;
```

子程序结构（4/5）

- 函数(function)
  - 格式：
    - 函数说明：FUNCTION <函数名>
      ([对象种类] 参数名[,参数名…]:[in] 数据类型;
      [对象种类] 参数名[,参数名…]:[in] 数据类型;
      ……) RETURN 数据类型;
    - 函数定义：FUNCTION <函数名> (参数表) RETURN 数据类型 IS
      [说明部分]；
      BIGIN
      <顺序语句>;
      RETURN (表达式）；
      END [FUNCTION] <函数名> ;

必不可少！

子程序结构（5/5）

- 参数种类：常量和信号；
- 参数方式：
  - 输入in：默认参数种类为常量；
- 说明部分：
  - 包括变量说明、常量说明和类型说明；
  - 函数中不能定义新的信号；
- 例：

```
FUNCTION func1 ( a,b,c : REAL ) RETURN REAL;
FUNCTION  "*"  ( a ,b : INTEGER ) RETURN INTEGER;
```

【例】

```
PROCEDURE  prg1
    (VARIABLE value:INOUT BIT_VECTOR(0 TO 3)) IS
BEGIN
 CASE value IS
  WHEN "0000" => value: "0101" ;
  WHEN "0101" => value: "0000" ;
  WHEN OTHERS => value: "1111" ;
 END CASE ;
END PROCEDURE prg1 ;
```

```
FUNCTION  max( a,b : IN STD_LOGIC_VECTOR)
    RETURN STD_LOGIC_VECTOR IS
 BEGIN
   IF a > b THEN   RETURN  a;
     ELSE          RETURN  b;
   END IF;
 END FUNCTION max;
```

重载

- 指同名的子程序，以用不同的数据类型作为子程序参数而定义多次；
- VHDL编译器会选择相适应的一个子程序；
- VHDL中的重载对象
  - 子过程，函数，运算符，可枚举数据类型等；
- 例
  - function "+"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
  - function "+"(L: SIGNED; R: SIGNED) return SIGNED;
  - function "+"(L: UNSIGNED; R: SIGNED) return SIGNED;

VHDL的预定义属性（1/15）

- 属性是指关于设计实体、结构体、类型、信号等项目的指定特征；
- 利用属性可以使VHDL源代码更加简明扼要，易于理解；
- VHDL提供五类属性：
  - 值类属性
  - 函数类属性
  - 信号类属性
  - 数据类型类属性
  - 数据范围类属性

VHDL的预定义属性（2/15）

- 值类属性
  - 主要用于返回常用数据类型、数组或是块的有关值；
    - 例如：返回数组长度，数据类型的上下界等；
  - 常用数据类型的值类属性
    - **‘left**: 返回一个数据类型或子类型最左边的值；
    - **‘right**:返回一个数据类型或子类型最右边的值；
    - **‘high**:返回一个数据类型或子类型最大值；
    - **‘low**:返回一个数据类型或子类型最小值；

VHDL的预定义属性（3/15）

- 数组的值类属性（‘length）
  - 功能: 返回限定性数组的长度值，即数组中元素的个数；
- 块的值类属性
  - **‘behavior**: 对于不含有任何元件例化语句的块语句或结构体，返回**‘true’**;
  - **‘structure**:对于只含有元件例化语句的块语句或结构体，返回**‘true’**;

常用数据类型的值类属性

```
Type number is integer range 0 to 9;
Type week is (mon, tue, wed, thu, fri, sat, sun);
Type word is array (15 downto 0) of std_logic;
……
```

Number'left=0;　　We　　Word'left=15;

Number'right=9;　　We　　Word'right=0;

Number'high=9;　　Week'high=sun;　　Word'high=15;

Number'low=0;　　mon;　　Word'low=0;

在递减区间中
'left='high;
'right='low;

在递增区间中
'left='low;
'right='high;

数组的值类属性

```
TYPE arry1 ARRAY (0 TO 7) OF BIT ;
VARIABLE wth: INTEGER;
...
wth1: =arry1'LENGTH; -- wth1 = 8
```

VHDL的预定义属性（4/15）

- 函数类属性
  - 指属性以函数的形式返回有关数据类型、数组或是信号的信息；
  - 属性根据输入的自变量值去执行函数，返回一个相应的值；

VHDL的预定义属性（5/15）

- 数据类型属性函数
  - 主要用来得到数据类型的各种相关信息；共六种；
  - **‘pos(数据值)**：返回数据类型定义中该数据值的位置序号；
  - **‘val(位置序号)**：返回数据类型定义中该位置序号处的值；
  - **‘succ(数据值)**：返回数据类型定义中该数据值的下一个值；
  - **‘pred(数据值)**：返回数据类型定义中该数据值的前一个值；

VHDL的预定义属性（6/15）

- ‘leftof(数据值): 返回数据类型定义中该数据值左边的一个值；
- ‘rightof(数据值): 返回数据类型定义中该数据值右边的一个值；
- 对于递增区间：
  - ‘succ(数据值)=’rightof(数据值);
  - ‘pred(数据值)=’leftof(数据值);
- 对于递减区间：
  - ‘succ(数据值)= ’leftof(数据值);
  - ‘pred(数据值)= ’rightof(数据值);

VHDL的预定义属性（7/15）

- 数组属性函数
  - **‘left(n)**:得到索引号为n的区间的左端位置号；
  - **‘right(n)**:得到索引号为n的区间的右端位置号；
  - **‘high(n)**:得到索引号为n的区间的高端位置号；
  - **‘low(n)**:得到索引号为n的区间的低端位置号；
  - 当索引号取缺省值时，该函数代表对一维区间进行操作；
  - 功能：用来返回数组的边界；

VHDL的预定义属性（8/15）

- 信号属性函数
  - 用来得到有关信号的行为功能信息；
  - 信号‘**event**: 当前的一个相当小的时间间隔内有信号事件发生，则返回’true’,否则返回‘false’;
  - 信号‘**active**:当前的一个相当小的时间间隔内信号活跃，则返回’true’,否则返回‘false’;
  - 信号‘**last_value**: 返回该信号在最近一个事件发生以前的值；
  - 信号‘**last_event**: 返回该信号从前一个事件发生到现在的时间值；
  - 信号‘**last_value**: 返回该信号从前一次活跃到现在的时间值；

VHDL的预定义属性（9/15）

- 信号事件发生（event）：
  - 信号从‘0’到‘1’或从‘1’到‘0’的一次变化为一次信号事件；
- 信号活跃（active）：
  - 信号值的任何变化都是信号活跃；
  - 包括所有信号事件，以及信号从‘0’到‘0’或从‘1’到‘1’的变化；
- 例：
  - **if (clk’event and clk=‘1’ and (clk’last_value=‘0’)) then**

VHDL的预定义属性（10/15）

- 信号类属性
  - 是根据所加属性的信号去建立一个新的信号，
  - 利用该类属性得到的相关信息类似于某些函数类属性所得到的信息；
    - 例：not(clk’stable) 与 clk’event……
  - 但是，该类属性可以用于正常信号能用的任何场合；
    - 包括用在敏感信号表中；

VHDL的预定义属性（11/15）

- 共四种属性
  - 信号‘delayed[(t)] ：可得到一个与所加属性的信号同类型的延迟信号，延迟时间为t;
    - 当t=0时，延迟时间为一个模拟周期；
    - 该属性可以用来检查信号保持时间；
  - 信号‘stable[(t)]: 当所加属性的信号在时间t内没有事件发生，则返回’true‘；否则返回’false‘;
    - 当t=0时，可得到与‘event属性相反的值；
    - 注意：语句**“NOT(clock’STABLE AND clock =‘1’)** ”的表达方式是不可综合的；

VHDL的预定义属性（12/15）

- 信号‘quiet[(t)]：当所加属性的信号在时间t内不活跃，则返回’true‘；否则返回’false‘；
  - 当t=0时，判断时间为一个模拟周期；
  - 典型应用是用来对中断处理优先机制进行建模
- 信号‘transaction:当所加属性的信号活跃时，信号将对前一值进行翻转；

VHDL的预定义属性（13/15）

- 数据类型属性
  - 用来得到所加属性的数据类型的基本类型
    - 只能作为另一种值类或函数类属性的前缀；
  - 格式：数据类型‘base：
  - 例：

```
Type week is (mon, tue, wed, thu, fri, sat, sun);
SubType job_day is week range mon to fri;
……
Week’right=sun;
Job_day’base’right=sun;
Job_day’right=fri;
```

VHDL的预定义属性（14/15）

- 数据范围类属性
  - 用来返回数据的区间范围；
  - 仅用于数组类型中的限定性数组；
  - 共两种属性：
    - 数组‘range[(n)]:得到索引号为n的区间范围；
    - 数组‘reverse_range[(n)]:得到索引号为n的区间的逆序范围；

```
SIGNAL range1 : IN STD_LOGIC_VECTOR (0 TO 7) ;
FOR i IN range1'RANGE LOOP; --0 to 7
FOR i IN range1‘REVERSE_RANGE LOOP; --7 DOWNTO 0
```

VHDL的预定义属性（15/15）

- 用户自定义属性
  - 可以被附加到实体、结构体、过程、变量、信号、标号等；
  - 格式：**ATTRIBUTE 属性名 : 数据类型;**
    ATTRIBUTE 属性名 OF 对象名 : 对象类型IS 值;

库、程序包、配置（1/7）

- 在VHDL中除了设计实体和结构体外，还有三个可以进行独立编译的源设计单元：
  - 库、程序包和配置；
- 库：用来存放已经编译的实体、结构体、程序包和配置；
  - 在MAX+plusII中，每个库对应一个文件夹；
  - 说明格式：library <库名>;

    USE 库名.程序包名.项目名(或 ALL)；

库、程序包、配置（2/7）

- 设计库：对当前的设计永远可见，不需要在代码开头部分进行说明；
  - Std库：VHDL的标准库，包括程序包standard和textio;
    - 注意：使用程序包textio时，需要声明；
  - Work库：使用者的当前工作路径；编译VHDL代码时，默认其保存在work库中；
  - 实际上所有VHDL源代码的开头部分都隐含了以下代码：

```
Library work;
Library std;
Use std.standard.all;
```

库、程序包、配置（3/7）

- 资源库：用来存放常规元件和常用模块的库，使用的时候需要进行声明；
  - 资源库内容与厂商直接相关；
  - Ieee库：目前使用最多，应用最广的资源库；
    - 包括：程序包std_logic_1164,numeric_bit, numeric_std, math_complex和math_real……；
  - 用户自定义库：设计人员可以建立自己的资源库；用来存放设计中所共用的一些程序包；
    - 使用自己定义的资源库时，需要进行声明；

引用库和程序包的例子

```
Library ieee;
Use ieee.all;
Entity and2 is
        port ( a,b: in std_logic_1164.std_logic;
                c: out std_logic_1164.std_logic);
```

```
Library ieee;
Use ieee.std_logic_1164.all;
Entity and2 is
        port ( a,b: in std_logic;
                c: out std_logic);
```

最常用!

```
Library ieee;
Use ieee.std_logic_1164.std_logic;
Entity and2 is
        port ( a,b: in std_logic;
                c: out std_logic);
```

库、程序包、配置（4/7）

- 程序包：
  - 用来存放各个设计都能共享的数据类型、子程序说明、属性说明和元件说明等部分；
  - 使用时，用use语句声明；
  - 一个程序包由两个部分组成：
    - 程序包说明部分；
    - 程序包包体部分；

库、程序包、配置（5/7）

- 程序包说明部分：
  - 主要对数据类型、子程序、常量、元件、属性及属性指定等进行说明；
  - 格式：

```
PACKAGE   <程序包名>   IS

        <程序包首说明部分>;

END [package] [<程序包名>];
```

  - 例：程序包std_logic_1164的程序包说明部分，**ieee**目录下**std1164.vhd**文件

库、程序包、配置（6/7）

- 程序包包体部分：
  - 由说明部分指定的函数和过程和程序体组成，即用来规定程序包的实际功能；
  - 描述方法与结构体的描述方式相同；
  - 格式：

```
PACKAGE BODY  <程序包名>  IS

  <程序包体说明部分以及包体内部数据对象说明>;

END [package body][<程序包名>];
```

  - 例：程序包std_logic_1164的程序包包体部分，**ieee**目录下**std1164b.vhd**文件

库、程序包、配置（7/7）

- 配置：用于描述各种设计实体和元件之间的连接关系以及设计实体和结构体之间的连接关系；
  - 一般格式：

```
CONFIGURATION <配置名> OF <实体名> IS
        for <选配结构体名>
              [<配置说明>;]
        end for;
END [<配置名>];
```

VHDL结构体的描述方式（1/3）

- 结构体描述的是设计的行为和结构，即描述一个设计实体的功能：
  - 实体硬件的结构；
  - 硬件的类型和功能；
  - 元件的互连关系；
  - 信号的传输和变换以及动态行为；
- 结构体描述的方式：
  - 行为描述方式；
  - 数据流描述方式；
  - 结构描述方式；

VHDL结构体的描述方式（2/3）

- 行为描述方式
  - 抽象度最高，无需知道具体电路的结构；
  - 只需要描述清楚输入与输出的行为；
  - 不需要关注设计功能的门级实现；
  - 难于进行逻辑综合；
- 数据流描述方式
  - 针对从信号到信号的数据流的路径形式进行描述；
  - 需要设计者对设计实体的功能实现及内部电路有一定的了解；
  - 该描述容易进行逻辑综合；

VHDL结构体的描述方式（3/3）

- 结构描述方式
  - 在多层次的设计中，通过调用库中的元件或已设计好的模块来完成设计实体功能的描述；
  - 描述只表示元件（或模块）之间的互连；
  - 引用元件时，要现在结构体说明部分进行元件的说明，然后在使用元件时进行例化；
  - 大型设计的首选方案；
- 混合方式
  - 上述三种方式组合使用；

行为描述

```
library ieee;
use ieee.std_logic_1164.all;
entity adder is
        port (a, b, cin: in std_logic;
                co, s: out std_logic);
end adder;

architecture behave of adder is
begin
  process (a,b,cin)
    variable n: integer range 0 to 3;
    constant s_vector:
std_logic_vector(0 to 3):="0101";
    constant co_vector:
std_logic_vector(0 to 3):="0011";
begin
    n:=0;
    if (a='1') then
        n:=n+1;
    end if;
    if (b='1') then
        n:=n+1;
    end if;
    if (cin='1') then
        n:=n+1;
    end if;
    s<=s_vector(n);
    co<=co_vector(n);
  end process;
end behave;
```

注意定义范围

数据流描述

```
library ieee;
use ieee.std_logic_1164.all;

entity adder is
        port (a,b,cin: in std_logic;
                co, s: out std_logic);
end adder;

architecture dataflow of adder is
        signal tmp1,tmp2: std_logic
begin
        tmp1<=a xor b;
        tmp2<=tmp1 and cin;
        s<=tmp1 xor cin;
        co<=tmp2 or (a and b);
end dataflow;
```

按照逻辑表达式进行描述；不适合大型设计。

结构描述

```
library ieee;
use ieee.std_logic_1164.all;
entity adder is
  port (a, b, cin: in std_logic;
        co, s: out std_logic);
end adder;
architecture structure of adder is
  component and2
    port (a, b: in std_logic;
          c:out std_logic);
  end component;
  component or2
    port (a, b: in std_logic;
          c:out std_logic);
  end component;
  component xor2
    port (a, b: in std_logic;
          c:out std_logic);
  end component;
```

- 该描述方式最为常用；
- 在已有的设计成果基础上，进行设计；
- 大型设计的首选方案；

```
  signal tmp1,tmp2,tmp3: std_logic;
begin
  u1: xor2 port map (a,b,tmp1);
  u2: and2 port map (tmp1,cin,tmp2);
  u3: xor2 port map (tmp1,cin,s);
  u4: and2 port map (a,b,tmp3);
  u5: or2 port map (tmp2,tmp3,co);
end structure;
```

混合描述

- 较为常用的方式；
- 形式灵活多样；

```
library ieee;
use ieee.std_logic_1164.all;
entity adder is
  port (a, b, cin: in std_logic;
        co, s: out std_logic);
end adder;
architecture mixed of adder is
  component or2
    port (a, b: in std_logic;
          c:out std_logic);
  end component;
  component xor2
    port (a, b: in std_logic;
          c:out std_logic);
  end component;
signal tmp1,tmp2,tmp3: std_logic;
begin
  tmp3<=a and b;
  tmp2<=tmp1 and cin;
  u1: xor2 port map (a,b,tmp1);
  u2: xor2 port map (tmp1,cin,s);
  u3: or2 port map (tmp2,tmp3,co);
end mixed;
```

END