第三章 数据类型

要求设计实体中的每一个常数、信号、变量、函数以及设定的各种参量都必须具有确定的数据类型，并且相同数据类型的量才能互相传递和作用。

- 预定义的数据类型
- 用户定义的数据类型
- 子类型
- 数组
- 端口数组
- 记录类型
- 有符号数和无符号数
- 数据类型转换

3.1 预定义的数据类型

指在IEEE 1076和IEEE 1164标准中预先定义的一系列数据类型，可以在包集/库中找到。

- std库的standard包集：BIT、BOOLEAN、INTEGER、REAL数据类型；
- ieee库的std_logic_1164包集：STD_LOGIC、STD_ULOGIC数据类型；
- ieee库的std_logic_arith包集：SIGNED、UNSIGNED数据类型;数据类型转换函数conv_integer(p),conv_unsigned(p,b)等；
- ieee库的std_logic_signed和std_logic_unsigned包集：一些函数，将STD_LOGIC_VECTOR类型数据进行类似SIGNED、UNSIGNED类型数据的运算；

1) BIT（位，表示一位的信号值，位值为‘0’或‘1’）和BIT_VECTOR（位矢量，表示一组位数据）。

- 声明：

```
SIGNAL X: BIT;
SIGNAL Y: BIT_VECTOR(3 DOWNTO 0);
SIGNAL W: BIT_VECTOR(0 DOWNTO 3);
```

注意：最高位MSB（Most Significant Bit）的顺序！

- 赋值：

```
X<='1';      -----单引号！
Y<="1001";   -----双引号！
```

2）STD_LOGIC和STD_LOGIC_VECTOR：

这两者是IEEE 1164标准中引入的8逻辑值系统。

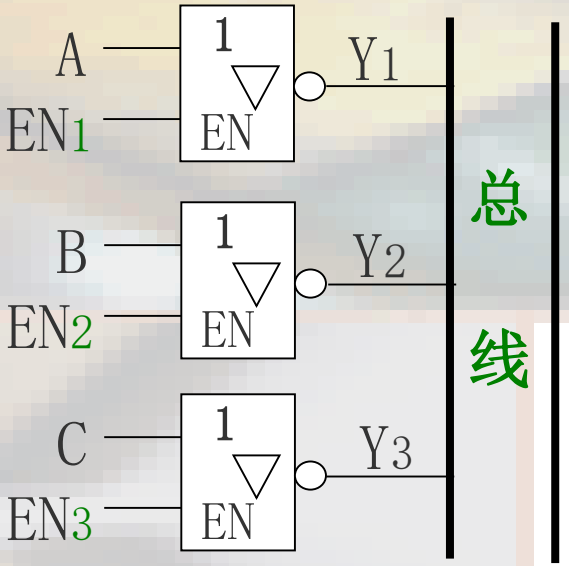
std_logic_vector类型是由 std_logic 构成的数组。定义如下：

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

赋值的原则：相同位宽，相同数据类型。

定义8种数字逻辑值的原因： 由std_logic 类型代替 bit 类型可以完成电子系统的精确模拟，并可实现常见的三态总线电路。

两个或以上数字逻辑电路的输出端连接到同一个节点时（称为“线与”现象！），节点的电平该如何取值？典型案例：总线！



节点的电平取值取决于：

- 两者或多者当前的输出电平值；
- 两者的驱动能力强弱。

驱动能力强的电路可以将节点电平强行拉高或拉低，因此需建立多值逻辑系统加以细分。

X：‘强’不确定值；　　W：‘弱’不确定值；

0：‘强’0；　　L：‘弱’0；

1：‘强’1；　　H：‘弱’1；

Z：高阻态（三态缓冲器，常用于总线设计）

-：不可能出现的情况

多个输出连接到同一个节点上时，节点的电平取值：

	X	0	1	Z	W	L	H	-
X	X	X	X	X	X	X	X	X
0	X	0	X	0	0	0	0	X
1	X	X	1	1	1	1	1	X
Z	X	0	1	Z	W	L	H	X
W	X	0	1	W	W	L	H	X
L	X	0	1	L	W	L	W	X
H	X	0	1	H	W	W	H	X
-	X	X	X	X	X	X	X	X

4

8逻辑值系统数值关系表

数值关系归纳：

- **X或-与其它数值连接时，最终电平取值均为X；**
- **Z与其它数值连接时，最终电平取值均为其它数值；**
- **与X类似，W与L/H数值连接时，最终电平取值均为W；**
- **0与1、L与H连接时，最终电平取值分别为X、W；**

STD_LOGIC_VECTOR类型数据的算术运算操作

- STD_LOGIC_VECTOR类型数据不能直接进行算术运算。通过声明ieee库中的std_signed和std_logic_unsigned这两个包集，该类型数据即可进行算术运算。
- 例：

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
signal a,b: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
signal c：  OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        c<=a+b;
```

3）STD_ULOGIC和STD_ULOGIC_VECTOR:

比STD_LOGIC类型多引入了一个逻辑值‘U’，代表初始不定值。但没有指定两个STD_ULOGIC信号连接到同一个节点上发生冲突后的逻辑值，因此要避免两个输出信号直接进行连接的情况。

4）布尔类型（boolean）

布尔量具有两种状态：false 和 true

常用于逻辑函数，如相等（=）、比较（<）等中作逻辑比较。 如，bit 值转化成boolean值：

```
boolean_var := (bit_var = '1');
```

5）字符（CHARACTER）：用单引号将字符括起来。

```
variable character_var : character;
... ...
Character_var : = 'A';
```

6）整数（integer）

integer 表示所有正的和负的整数。硬件实现时，利用32位的位矢量来表示。可实现的整数范围为：

$-(2^{31}-1)$ to $(2^{31}-1)$

VHDL综合器要求对具体的整数作出范围限定，否则无法综合成硬件电路。

如：signal s : integer range 0 to 15;

信号 s 的取值范围是0-15，可用4位二进制数表示，因此 s 将被综合成由四条信号线构成的信号。

7）自然数（natural）和正整数（positive）

natural是integer的子类型，表示非负整数。

positive是integer的子类型，表示正整数。

定义如下：

```
subtype natural is integer range 0 to
    integer'high;
subtype positive is integer range 1 to
    integer'high;
```

8）实数（REAL）

或称浮点数

取值范围：-1.0E38 - +1.0E38

实数类型仅能用于VHDL仿真器，一般综合器不支持。

9）物理量字符（Physical literal）：

时间、电压等，可以仿真，但不可综合（即综合库中没有直接可以调用的器件）。

由整数和物理单位组成

如：55 ms，20 ns

10）SIGNED(有符号数)和UNSIGNED（无符号数）：

ieee库std_logic_arith包集中定义的数据类型，只能表示大于等于0的数，能够支持算术运算、比较运算，但不支持逻辑运算。

只有在代码开始部分声明ieee库中的包集std_logic_arith，才能使用有符号数和无符号数。

有符号数和无符号数的语法结构与STD_LOGIC_VECTOR相似，与整数不同，例如：

```
SIGNAL X: SIGNED(7 DOWNTO 0);
SIGNAL Y: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL Z: INTEGER RANGE 0 TO 255;
```

- 例：signed和unsigned数的合法与非法操作：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;--必须声明该包集才能使用signed和unsigned数。
signal a,b:IN SIGNED(7 DOWNTO 0);
signal x: OUT SIGNED(7 DOWNTO 0);
x<=a+b;  --合法（支持算术运算）
x<=a AND b;  --非法（不支持逻辑运算）
```

- 例：STD_LOGIC_VECTOR的合法与非法操作：

```
library ieee;
use ieee.std_logic_1164.all; --不必声明其它包集。
signal a,b:IN std_logic_vector(7 DOWNTO 0);
signal x: OUT std_logic_vector(7 DOWNTO 0);
x<=a+b;  --非法（不支持算术运算）
x<=a AND b;  --合法（支持逻辑运算）
```

注意：如果声明std_logic_signed和std_logic_unsigned两个包集，则STD_LOGIC_VECTOR类型的数据也可以进行算术运算。

- 例：STD_LOGIC_VECTOR的合法与非法操作：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
signal a,b:IN std_logic_vector(7 DOWNTO 0);
signal x: OUT std_logic_vector(7 DOWNTO 0);
x<=a+b;  --合法（支持算术运算）
x<=a AND b;  --合法（支持逻辑运算）
```

3.2 用户定义的数据类型

用类型定义语句TYPE实现用户自定义数据类型。

TYPE语句格式：

可选

type 数据类型名 is 数据类型定义 [of 基本数据类型];

例：type byte is array(7 downto 0) of bit;

variable addend : byte;

type week is (sun, mon, tue, wed, thu,
fri, sat);

1）用户定义的整数类型

用户定义的整数类型是标准包中整数类型的子范围。格式：

```
type 类型名称 is range 整数范围;
```

例：

```
type my_natural is range 0 to 9; ---用户定义的自然数类型;

type my_integer is range -3 to 3;---用户定义的整数类型;
```

2）枚举（enumerated）类型

枚举该类型的所有可能的值。格式：

type 类型名称 is (枚举文字{，枚举文字})；

如：type my_logic is(‘0’, ‘1’, ‘Z’);

type state is (idle,forward,backward, stop);--常用于有限状态机的定义。

type color is(blue,green,yellow, red);

variable hue : color;

hue := blue;

type bit_vector is array(natural range <>) of BIT;---range<>表示数据取值范围没有约束，natural range<>表示数据值约束在自然数范围内。

枚举类型的编码：

综合器自动实现枚举类型元素的编码，一般将第一个枚举量（最左边）编码为0，以后的依次加1。编码用位矢量表示 ，位矢量的长度将取所需表达的所有枚举元素的最小值。

如：type color is(blue,green,yellow,red);

编码为： blue=“00”;

green=“01”;

yellow=“10”;

red=“11”;

3.3 子类型

子类型是已定义的类型或子类型的一个子集。

格式：

```
subtype 子类型名 is 数据类型名[范围];
```

例：

bit_vector 类型定义如下：

```
type bit_vector is array (natural range <>)
    of bit;
```

如设计中只用16bit；可定义子类型如下：

```
subtype my_vector is bit_vector(0 to 15);
```

注：子类型与基（父）类型具有相同的操作符和子程序。可以直接进行赋值操作。

SUBTYPE语句格式：

subtype 子类型名 is 基本数据类型 约束范围;

例：

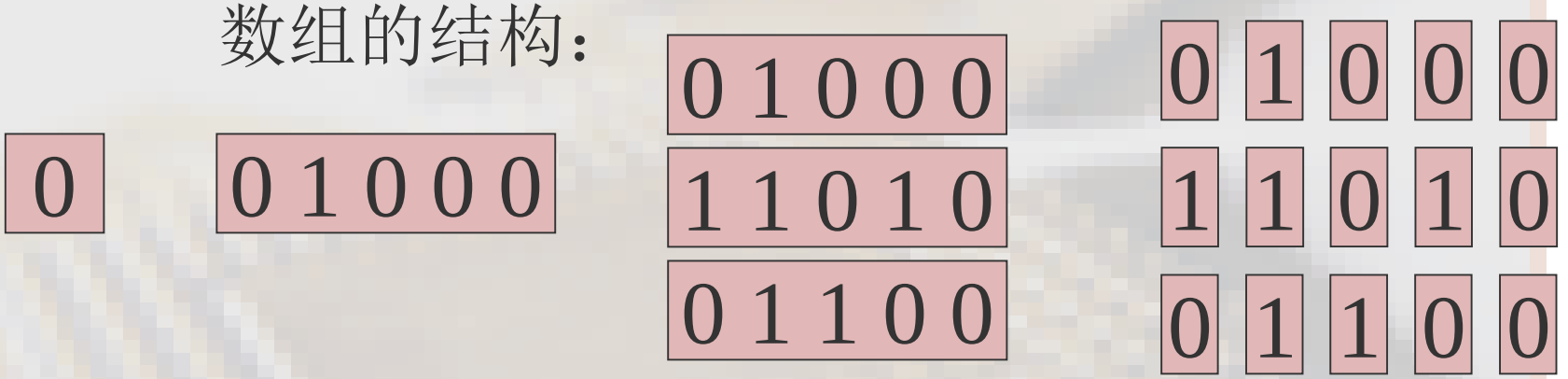
```
subtype digits is integer range 0 to 9;
```

由subtype 语句定义的数据类型称为子类型。

3.4 数组（ARRAY）

数组是将相同数据类型的数据集合在一起形成的一种新的数据类型。可以是1D、2D或1D*1D,更高维数的数组往往是不可综合（即综合库中没有直接可以调用的器件）的。

数组的结构：



a.标量　**b.1D数组**　矢量　**c.1D*1D**　矢量数组　**d.2D**　二维标量数组

VHDL中预定义的数据类型仅包括标量类型（单个位）和矢量类型（一维数组）两类，并没有预定义2D和1D*1D数组，用户可以自定义。定义的语法如下：

```
type type_name is array( specification ) of data_type;
```

数组类型对signal/variable/constant的声明的语法如下：

```
signal (constant/variable） signal_name: type_name
[:=initial_value];
```

可选

例子：

- 一种定义1D*1D数组的方法：

矢量

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;--1D数组;
TYPE matrix IS ARRAY (0 TO 3) OF row;    --1D*1D数组，矢量数组;
SIGNAL x: matrix;                        ---声明是1D*1D信号
```

- 另一种定义1D*1D数组的方法：

```
TYPE matrix IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
```

- 二维数组的定义方法：

标量

```
TYPE matrix2D IS ARRAY(0 TO 3,7 DOWNTO 0) OF STD_LOGIC;
```

例：数组的初始化：

CONSTANT X: vector_array:=(”0011”,”1100”,”0101”);--常用于指令或数据ROM设计中。

signal Y: vector_array2:=(‘1’,’0’,’0’,’1’);

variable Z: vector_array3: =((‘0’,’1’,’1’,’0’) ,(‘1’,’0’,’1’,’1’));

例：合法与非法的数组赋值：

```
TYPE row IS ARRAY(7 DOWNTO 0) OF STD_LOGIC;
TYPE array1 IS ARRAY(0 TO 3) OF row;
TYPE array2 IS ARRAY(0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
TYPE array3 IS ARRAY(0 TO 3,7 DOWNTO 0) OF STD_LOGIC;
```

```
SIGNAL x: row;
SIGNAL y: array1;
SIGNAL v: array2;
SIGNAL w: array3;
-----------------------------------------
```

x<=v(1);　　--非法，类型不匹配，x是row类型，而v(1)是STD_LOGIC_VECTOR类型的。

x<=w(2);　　--非法，w必须带两个索引值；

x<=w(2,2 DOWNTO 0);　--非法，X是row类型的，而右侧是std_logic类型的。

v(0)<=w(2,2 DOWNTO 0); --非法，v（0）是std_logic_vector类型的，右侧是std_logic类型的，数据类型不匹配。

w(1,5 DOWNTO 1)<=v(2)(4 DOWNTO 0);--非法，类型不匹配。

3.5 端口数组,例：

```
----------------------包集----------------------
library ieee;
use ieee.std_logic_1164.all;
------------------------------
PACKAGE my_data_types IS
   TYPE vector_array IS ARRAY(NATURAL RANGE<>) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
END my_data_types;
--------------------主代码----------------------
library ieee;
use ieee.std_logic_1164.all;
use work.my_data_types.all;    ---用户定义的包集
-----------------------------
ENTITY mux IS
   PORT (inp: IN vector_array(0 TO 3);
         ....      );
END mux;

..............
```

ENTITY中不允许使用TYPE进行类型定义，须在包集中自定义。

----------包集----------

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

PACKAGE my_data_types IS
    CONSTANT b:  INTEGER:=7;
    TYPE vector_array IS ARRAY(NATURAL RANGE<>) OF
                         STD_LOGIC_VECTOR(b DOWNTO 0);
END my_data_types;
-----------------------------
```

包含常量的声明

3.6 记录类型

记录是不同类型的名称域的集合，而ARRAY只能包含相同类型的数据。

格式如下：

```
type 记录类型名 is record
        元素名：数据类型名;
        元素名：数据类型名;
          ⋮
end record;
```

访问记录体元素的方式：记录体名.元素名

例：

```
constant len:integer:= 8 ;
subtype byte_vec is bit_vector(len-1 downto 0);
type byte_and_ix  is  record
        byte : byte_vec;
        ix : integer range 0 to len;
end  record ;
signal  x, y, z : byte_and_ix ;
signal  data : byte_vec ;
signal  num : integer ;
                .......
x.byte <= “11110000” ;
x.ix <= 2 ;
data <= y.byte ;
num <= y.ix ;
z <= x ;
```

3.7 数据类型转换

VHDL是一种强类型语言，不同类型的数据对象必须经过类型转换，才能相互操作。两种实现数据类型转换的常见方法：

1）写一段专用于数据类型转换的VHDL代码

2）调用包集中预定义的数据类型转换函数，如包集std_logic_1164。

例：不同类型数据的合法与非法操作

```
TYPE long IS INTEGER RANGE -100 TO +100;
TYPE short IS INTEGER RANGE -10 TO +10;
SIGNAL x: short;
SIGNAL y: long;

y <= 2*x+5 ;    ---非法（数据类型不匹配，虽然都是INTEGER的子类型！）
y <= long(2*x+5);  ---合法（运算结果已经强制转换成long类型。）
```

ieee.std_logic_arith中提供了多种数据类型转换函数：

不包括std_logic_vector类型，
如有需要，须使用
std_logic_unsigned/signed包集

FUNCTION	Pass(arg, size)		Return
• CONV_INTEGER	INTEGER		INTEGER
• CONV_INTEGER	UNSIGNED		INTEGER
• CONV_INTEGER	SIGNED		INTEGER
• CONV_INTEGER	STD_ULOGIC		SMALL_INT;
• CONV_UNSIGNED	INTEGER,	INTEGER	UNSIGNED;
• CONV_UNSIGNED	UNSIGNED,	INTEGER	UNSIGNED;
• CONV_UNSIGNED	SIGNED,	INTEGER	UNSIGNED;
• CONV_UNSIGNED	STD_ULOGIC,	INTEGER	UNSIGNED;
• CONV_SIGNED	INTEGER,	INTEGER	SIGNED;
• CONV_SIGNED	UNSIGNED,	INTEGER	SIGNED;
• CONV_SIGNED	SIGNED,	INTEGER	SIGNED;
• CONV_SIGNED	STD_ULOGIC,	INTEGER	SIGNED;
• CONV_STD_LOGIC_VECTOR	INTEGER,	INTEGER	STD_LOGIC_VECTOR
• CONV_STD_LOGIC_VECTOR	UNSIGNED,	INTEGER	STD_LOGIC_VECTOR
• CONV_STD_LOGIC_VECTOR	SIGNED,	INTEGER	STD_LOGIC_VECTOR
• CONV_STD_LOGIC_VECTOR	STD_ULOGIC,	INTEGER	STD_LOGIC_VECTOR

例：数据类型转换

```
library ieee;
 use ieee.std_logic_1164.all;
 use ieee.std_logic_arith.all;
 .......
 signal a: IN unsigned(7 DOWNTO 0);
 signal b: IN unsigned(7 DOWNTO 0);
 signal y: OUT std_logic_vector(7 DOWNTO 0);
 .......
 y<=CONV_STD_LOGIC_VECTOR((a+b),8);
```

3.8 可综合的数据类型

数据类型	可综合的数值
BIT, BIT_VECTOR	‘0’, ‘1’
STD_LOGIC, STD_LOGIC_VECTOR	‘X’, ‘0’, ‘1’, ‘Z’, 不是全部的8值都可综合的；另， 在不需要’ X’,’Z’两种取值时可用BIT类型混用。
STD_ULOGIC, STD_ULOGIC_VECTOR	‘X’, ‘0’, ‘1’, ‘Z’, 不是全部的8值都可综合的；另， 在不需要’ X’,’Z’两种取值时可用BIT类型混用。
BOOLEAN	True, False
NATURAL / UNSIGNED	0到+2 147 483 647
INTEGER / SIGNED	-2 147 483 647到+2 147 483 647
用户自定义整型	INTEGER的子集
用户自定义枚举类型	根据用户自定义进行编码得到
SUBTYPE	任何预定义或用户自定义类型的子集
ARRAY	上述任一种类型数据的集合
RECORD	上述多种类型数据的集合

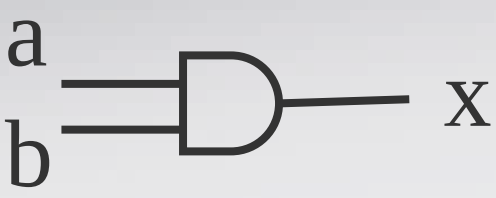
- 例子：常用数据类型的声明与赋值

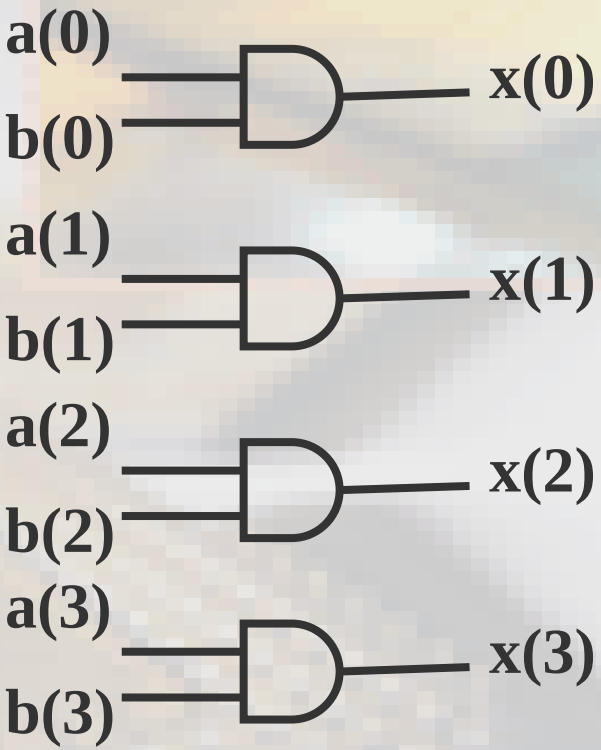
```
signal a: BIT;
signal b: BIT_VECTOR(7 DOWNTO 0);
signal c: STD_LOGIC;
signal d: STD_LOGIC_VECTOR(7 DOWNTO 0);
signal e: INTEGER RANGE 0 TO 255;

a<=b(5);                                    a<=c;
b(0)<=a;                                    b<=d;    类型不匹配
c<=d(5);                                    e<=b;
d(0)<=c;                                    e<=d;
b<=(7=>'0',1=>'1',OTHERS=>'1');
C<='Z';
```

例子： 单个位和位矢量

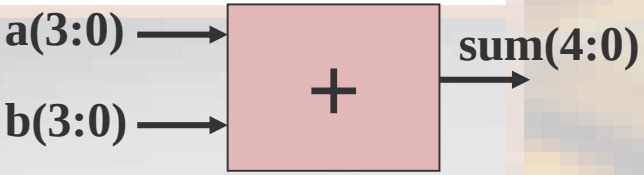
```
ENTITY and2 IS
  PORT (a,b: IN BIT;
        x: OUT BIT);
END and2;
architecture and2 of and2 is
BEGIN
   x<=a AND b;
END and2;
```

```
ENTITY and2 IS
  PORT (
      a,b : in bit_vector (0 TO 3);
      x:out bit_vector(0 TO 3));
  END and2;
architecture and2 of and2 is
BEGIN
    x<=a AND b;
END and2;
```


a
b
x


a(0)
b(0)
x(0)
a(1)
b(1)
x(1)
a(2)
b(2)
x(2)
a(3)
b(3)
x(3)

例子： 4位加法器



4位加法器

```
library ieee;
use ieee.std_logic_1164.all;
use iee.std_logic_arith.all;
entity adder1 is
  port (a,b: in signed(3 downto 0);
       sum: out signed(4 downto 0) );
end adder1;
architecture adder1 of adder1 is
begin
   sum<=a + b;
end adder1;
```

```
library ieee;
use ieee.std_logic_1164.all;
use iee.std_logic_arith.all;
entity adder2 is
  port (a,b: in signed(3 downto 0);
   sum: out integer range -16 to 15) );
end adder2;
architecture adder2 of adder2 is
begin
    sum<=conv_integer (a + b);
end adder2;
```

补充：VHDL文字规则

1、数字型文字

1）整数文字：十进制整数

如：5，678，156E2（=15600），

45_234_287 （=45234287）

2）实数文字：带小数的十进制数

如：23.34，2.0，44.99E-2（=0.4499）

8_867_551.23_909（8867551.23909）

3）以数制基数表示的文字

格式：

基数#数字文字#E指数

如：10#170#（=170）

2#1111_1110#（=254）

16#E#E1（=2#1110_0000# =224）

或：（=14×16=224）

16#F.01#E+2

（=(15+1/(16×16))×16×16=3841.00）

4）物理量文字

如：60 s、100 m、 177 mA

注：整数可综合实现；

实数一般不可综合实现；

物理量不可综合实现；

2、字符串型文字（文字串和数字串）

按字符个数多少分为：

字符：用单引号引起来的ASCII字符，可以是数值，也可以是符号或字母。

如：‘A’，‘*’，‘Z’

字符串：用双引号引起来的一维字符数组

字符串分为：

1）文字字符串：“文字”

如：“ERROR”，

“XXXXXXXX”，

“ZZZZZZZZ”，--必须是大写

“X”，

“BOTH S AND Q EQUAL TO L”。

数位字符串：

称为位矢量，代表二进制、八进制、十六进制的数组。其位矢量的长度为等值的二进制数的位数。

格式：

基数符号 “数值”

其中基数符号有三种：

B：二进制基数符号。

O：八进制基数符号，每一个八进制数代表一个3位的二进制数。

X：十六进制基数符号，每一个十六进制数代表一个4位的二进制数。

如：

B“1_1101_1110”：二进制数数组，长度为9

O“34”：八进制数数组，长度为6

X“1AB”：十六进制数数组，长度为12

标识符

定义常数、变量、信号、端口、子程序或参数的名字。

基本标识符的要求（87标准）：

- 以英文字母开头；
- 不连续使用下划线“_”；
- 不以下划线“_”结尾；
- 由26个大小写英文字母、数字0-9及下划线“_”组成的字符串。

基本标识符中的英文字母不分大小写；

VHDL的保留字不能作为标识符使用。

合法标识符	不合法标识符
my_counter、	_Decoder_1、
Decoder_1、	2FFT、
FFT、	Sig_#N、
Sig_N、	Not-Ack、
Not_Ack、	ALL_RST_、
State0、	data__BUS、
entity1	return、
	entity

- 扩展标识符（93标准）：

以反斜杠来界定，免去了87标准中基本标识符的一些限制。

可以以数字打头，

允许使用VHDL保留字，

区分字母大小写等。

如：\74LS163\、 \Sig_#N\、

\entity\、\ENTITY\

- 注释符号： “--”，而不是“//”

第3章　课后思考题

1、VHDL中有哪些基本数据类型？有哪些是可以综合的数据类型？

2、理解STD_LOGIC和STD_LOGIC_VECTOR的8值逻辑系统。

3、包集中定义的数据类型转换函数。

4、表达式 c <= a + b 中，a、b和c的数据类型都是 std_logic_vector，是否能直接进行加法运算？说明原因和解决方法。

作业：　3.4、3.5