# 数字系统设计

郑海永

中国海洋大学 电子工程系

2014 年 4 月

# VHDL

# 目录

# 内容提要

# 内容提要

# ① 基本标识符

**规则**

- 英文字母（a-z，A-Z）、阿拉伯数字（0-9）和下划线（_）。
- 起始字符必须为英文字母。
- 下划线的前后必须有英文字母或者阿拉伯数字。
- 标识符不区分大小写。

**注意**

- VHDL 保留字（如ENTITY、ARCHITECTURE、BEGIN、END 等）不能用来命名用户声明项。
- 建议 VHDL 保留字统一大写（小写），用户声明标识符相反。
- 为保证 HDL 代码可移植性，建议不使用其他 HDL 语言保留字。
- 建议尽量采用统一且有意义的名称声明标识符。如full_adder。
- 建议采用统一的时钟信号和复位信号名称。如clk、reset、clk1 等。

# ① 基本标识符

## 规则

- 英文字母（a-z，A-Z）、阿拉伯数字（0-9）和下划线（_）。
- 起始字符必须为英文字母。
- 下划线的前后必须有英文字母或者阿拉伯数字。
- 标识符不区分大小写。

## 注意

- VHDL 保留字（如ENTITY、ARCHITECTURE、BEGIN、END 等）不能用来命名用户声明项。
- 建议 VHDL 保留字统一大写（小写），用户声明标识符相反。
- 为保证 HDL 代码可移植性，建议不使用其他 HDL 语言保留字。
- 建议尽量采用统一且有意义的名称声明标识符。如full_adder。
- 建议采用统一的时钟信号和复位信号名称。如clk、reset、clk1 等。

# ② 扩展标识符

VHDL'93

- 由一对反斜线之间的任何字符序列构成。
- 如果字符序列中还包含反斜线，则用双写的反斜线作为一个字符。
- 扩展标识符取消了基本标识符的所有限制：可以使用任何字符，以任何字符开头，区分大小写。
- 每一个扩展标识符与任何基本标识符都不相同，也不同于任何一个保留字。

```
1 \ENTITY\ \entity\
2 MUX  \MUX\  \mux\  \Mux\
3 \c:\\file\
```

# 内容提要

# 数据对象

常量 CONSTANT 常数类的对象能够用来储存某种指定类型的特定数值。
常量在仿真开始前赋值，仿真过程中常量的值不能改变。
对于子程序中声明的常量，在每次子程序调用时赋值。

变量 VARIABLE 变量类的对象也能够用来储存某种指定类型的一个数值。

信号 SIGNAL 信号类的对象用于存储一系列值，这些值可能是当前信号值，也可能是未来可能出现的信号值。

文件 FILE 文件类的对象包括一系列的值，这些值可以通过读写过程从文件里读写。

- 常量和变量就像高级编程语言（C）中的相应部分，信号可以看作是电路中的连线。
- 常量和变量通常用来对电路的行为建模，信号通常用来对连线和触发器建模，文件用来对主机环境中的文件建模。

# 数据对象

常量 CONSTANT 常数类的对象能够用来储存某种指定类型的特定数值。

变量 VARIABLE 变量类的对象也能够用来储存某种指定类型的一个数值。
可以采用不同的变量赋值语句在不同时间对变量赋值。

信号 SIGNAL 信号类的对象用于存储一系列值，这些值可能是当前信号值，也可能是未来可能出现的信号值。

文件 FILE 文件类的对象包括一系列的值，这些值可以通过读写过程从文件里读写。

- 常量和变量就像高级编程语言（C）中的相应部分，信号可以看作是电路中的连线。
- 常量和变量通常用来对电路的行为建模，信号通常用来对连线和触发器建模，文件用来对主机环境中的文件建模。

# 数据对象

常量 CONSTANT 常数类的对象能够用来储存某种指定类型的特定数值。

变量 VARIABLE 变量类的对象也能够用来储存某种指定类型的一个数值。

信号 SIGNAL 信号类的对象用于存储一系列值，这些值可能是当前信号值，也可能是未来可能出现的信号值。
可以采用信号赋值语句对信号赋予未来的值。

文件 FILE 文件类的对象包括一系列的值，这些值可以通过读写过程从文件里读写。

- 常量和变量就像高级编程语言（C）中的相应部分，信号可以看作是电路中的连线。
- 常量和变量通常用来对电路的行为建模，信号通常用来对连线和触发器建模，文件用来对主机环境中的文件建模。

# 数据对象

常量 CONSTANT 常数类的对象能够用来储存某种指定类型的特定数值。

变量 VARIABLE 变量类的对象也能够用来储存某种指定类型的一个数值。

信号 SIGNAL 信号类的对象用于存储一系列值，这些值可能是当前信号值，也可能是未来可能出现的信号值。

文件 FILE 文件类的对象包括一系列的值，这些值可以通过读写过程从文件里读写。

- 常量和变量就像高级编程语言（C）中的相应部分，信号可以看作是电路中的连线。
- 常量和变量通常用来对电路的行为建模，信号通常用来对连线和触发器建模，文件用来对主机环境中的文件建模。

# 数据对象

常量 CONSTANT 常数类的对象能够用来储存某种指定类型的特定数值。

变量 VARIABLE 变量类的对象也能够用来储存某种指定类型的一个数值。

信号 SIGNAL 信号类的对象用于存储一系列值，这些值可能是当前信号值，也可能是未来可能出现的信号值。

文件 FILE 文件类的对象包括一系列的值，这些值可以通过读写过程从文件里读写。

- 常量和变量就像高级编程语言（C）中的相应部分，信号可以看作是电路中的连线。
- 常量和变量通常用来对电路的行为建模，信号通常用来对连线和触发器建模，文件用来对主机环境中的文件建模。

# 数据对象

常量 CONSTANT 常数类的对象能够用来储存某种指定类型的特定数值。

变量 VARIABLE 变量类的对象也能够用来储存某种指定类型的一个数值。

信号 SIGNAL 信号类的对象用于存储一系列值，这些值可能是当前信号值，也可能是未来可能出现的信号值。

文件 FILE 文件类的对象包括一系列的值，这些值可以通过读写过程从文件里读写。

- 常量和变量就像高级编程语言（C）中的相应部分，信号可以看作是电路中的连线。
- 常量和变量通常用来对电路的行为建模，信号通常用来对连线和触发器建模，文件用来对主机环境中的文件建模。

# ① 常量声明

```
1 CONSTANT rise_time: TIME := 10 ns;
2 CONSTANT bus_width: INTEGER := 8;
```

**延迟常量** 只能出现在程序包声明中，相应程序体中必须出现带有相关值的完整的常量声明。

```
1 CONSTANT no_of_inputs: INTEGER;
```

# ① 常量声明

```
1 CONSTANT rise_time: TIME := 10 ns;
2 CONSTANT bus_width: INTEGER := 8;
```

**延迟常量** 只能出现在程序包声明中，相应程序体中必须出现带有相关值的完整的常量声明。

```
1 CONSTANT no_of_inputs: INTEGER;
```

# ② 变量声明

```
1 VARIABLE ctrl_status: Bit_Vector (10 DOWNTO 0);
2 VARIABLE sum: INTEGER RANGE 0 TO 100 := 10;
3 VARIABLE found, done: BOOLEAN;
```

- false 是BOOLEAN 预定义类型的最左值。
- 变量中每个元素的初值为相应的每种元素类型的初值（最左值）。

# ③ 信号声明

```
1 SIGNAL clock: Bit;
2 SIGNAL data_bus: Bit_Vector (0 TO 7);
3 SIGNAL gate_delay: TIME := 10 ns;
4 SIGNAL init_p: STD_LOGIC_VECTOR (7 DOWNTO 0) :=
5              (0 => '1', OTHERS => 'U');
```

- 0 是Bit 类型的最左值。

# ④ 文件声明

```
1 FILE filename: FILE_TYPE [[OPEN Model] IS string];
```

```
1 TYPE std_logic_file IS FILE OF STD_LOGIC_VECTOR;
2 TYPE bit_file IS FILE OF Bit_Vector;
3
4 FILE stimulus: TEXT OPEN READ_MODE IS "/usr/home/flyzhy/add.sti";
5 FILE vectors: bit_file IS "/usr/home/flyzhy/add.vec";
6 FILE pat1, pat2: std_logic_file;
```

# ④ 文件声明

```
FILE filename: FILE_TYPE [[OPEN Model] IS string];
```

```
TYPE std_logic_file IS FILE OF STD_LOGIC_VECTOR;
TYPE bit_file IS FILE OF Bit_Vector;

FILE stimulus: TEXT OPEN READ_MODE IS "/usr/home/flyzhy/add.sti";
FILE vectors: bit_file IS "/usr/home/flyzhy/add.vec";
FILE pat1, pat2: std_logic_file;
```

# 声明对象的其他方法

- 实体的端口。所有端口都是信号对象。
- 实体的类属。常量对象。
- 函数和过程的形式参数。函数参数是常量或信号，而过程参数可以是任何一种对象。
- 隐式声明。for ... loop 和for ... generate

```
1 FOR count IN 1 TO 10 LOOP
2   sum := sum + count;
3 END LOOP;
```

# 信号与变量

对象种类 { 名称 }: 对象类型 [ := 缺省值 ];

## 相同点

- 如果没有指定缺省值，默认值取该对象数据类型的最小值或最左值。

## 不同点

- 信号与硬件中互连元件端口的“连线”相对应。
  赋给信号的值必须是在某一事件发生并经过一段时间延迟之后，才能成为当前值。
- 变量在硬件中没有明确的对应物。
  赋给变量的值是立即发生的。
  变量是为了便于设计实体的行为描述而定义的暂存区只用于对数据的暂时存储，在设计实体的结构描述中不可见。

# 信号与变量

对象种类 { 名称 }: 对象类型 [ := 缺省值 ];

## 相同点

- 如果没有指定缺省值，默认值取该对象数据类型的最小值或最左值。

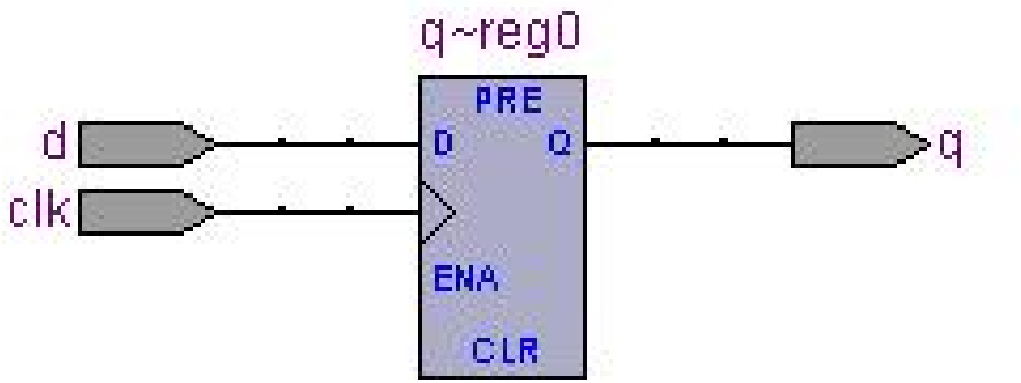
## 不同点

- 信号与硬件中互连元件端口的“连线”相对应。
  赋给信号的值必须是在某一事件发生并经过一段时间延迟之后，才能成为当前值。
- 变量在硬件中没有明确的对应物。
  赋给变量的值是立即发生的。
  变量是为了便于设计实体的行为描述而定义的暂存区只用于对数据的暂时存储，在设计实体的结构描述中不可见。

# D 型触发器——bhv1

```
1  ENTITY dff_1 IS
2    PORT (clk, d: IN Bit;
3          q: OUT Bit);
4  END dff_1;
5
6  ARCHITECTURE bhv1 OF dff_1 IS
7  BEGIN
8    d1: PROCESS(clk)
9    VARIABLE a: Bit;
10   BEGIN
11     IF clk'Event AND clk='1' THEN
12       a := d;
13       q <= a;
14     END IF;
15   END PROCESS d1;
16 END bhv1;
```

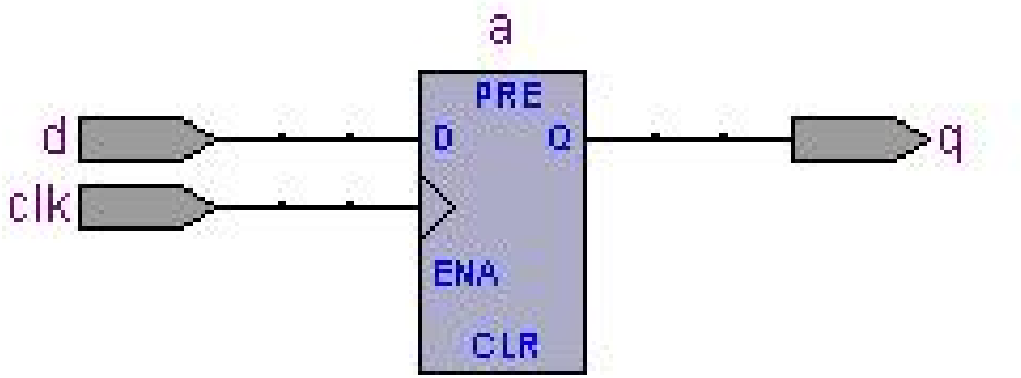
- 变量a 在硬件上没有对应实物存在，物理上综合不出元件。
- 通常用于临时保存数据值。


q~reg0
PRE
d
D
Q
q
clk
ENA
CLR

# D 型触发器——bhv2

```
1  ENTITY dff_2 IS
2    PORT (clk, d: IN Bit;
3          q: OUT Bit);
4  END dff_2;

6  ARCHITECTURE bhv2 OF dff_2 IS
7    SIGNAL a: Bit;
8  BEGIN
9    d2: PROCESS(clk)
10   BEGIN
11     IF clk'Event AND clk='1' THEN
12       a <= d;
13     END IF;
14   END PROCESS d2;
15   q <= a;
16 END bhv2;
```

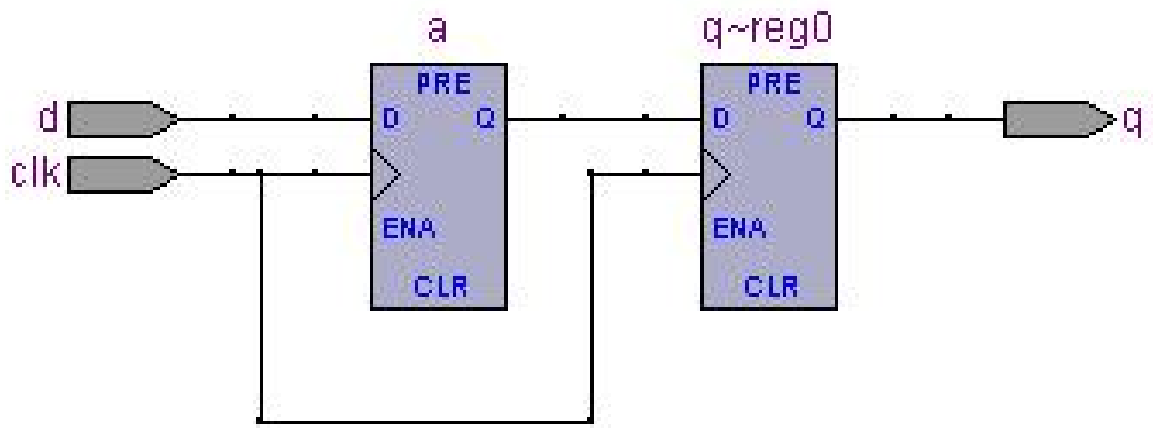
- 信号a 在硬件上是物理存在的。
- 信号赋值语句q <= a（缓冲器）与信号clk 无关。

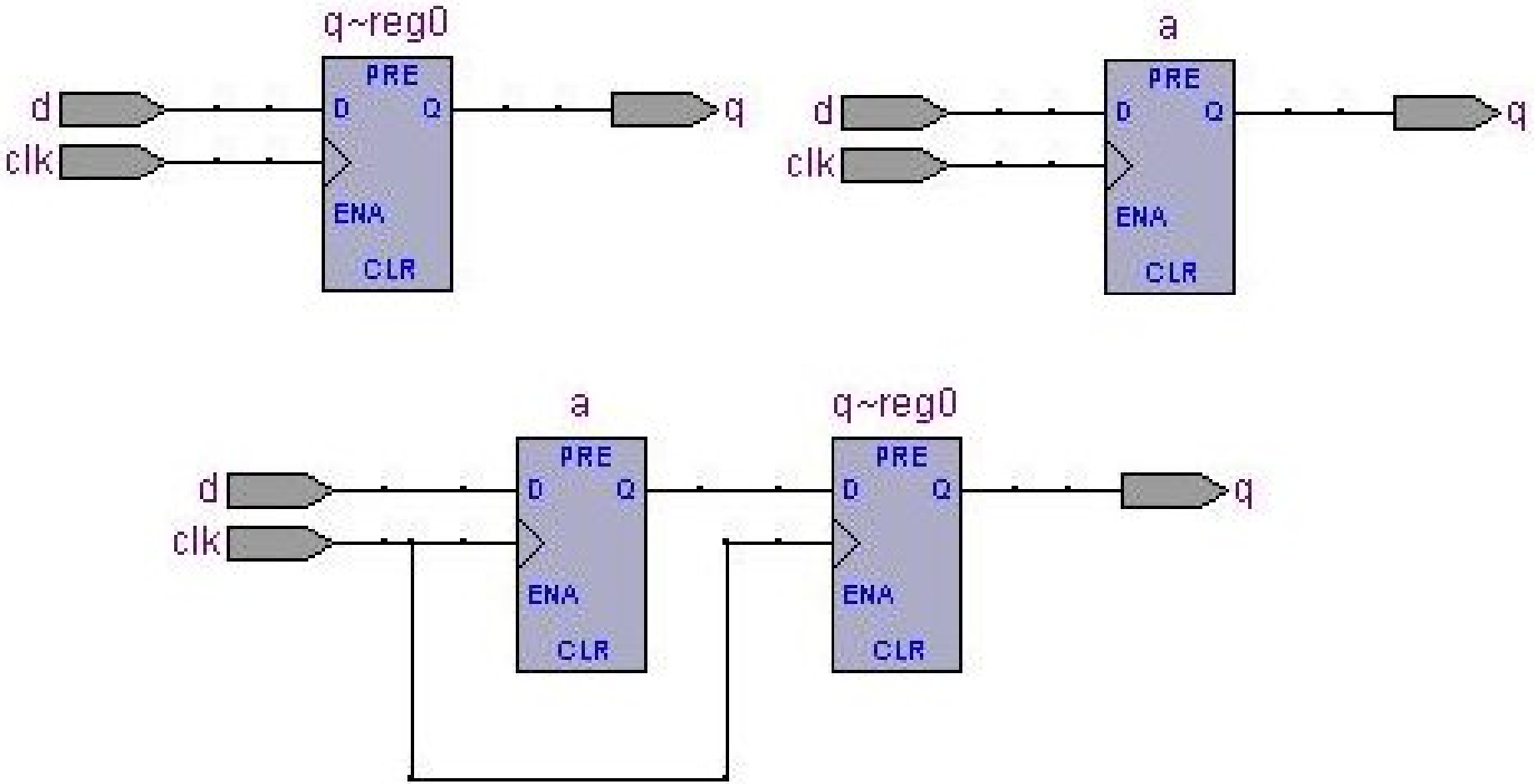

a
PRE
d
D
Q
q
clk
ENA
CLR

# D 型触发器——bhv3

```
1  ENTITY dff_3 IS
2    PORT (clk, d: IN Bit;
3          q: OUT Bit);
4  END dff_3;
5
6  ARCHITECTURE bhv3 OF dff_3 IS
7    SIGNAL a: Bit;
8  BEGIN
9    d3: PROCESS(clk)
10   BEGIN
11     IF clk'Event AND clk='1' THEN
12       a <= d;
13       q <= a;
14     END IF;
15   END PROCESS d3;
16 END bhv3;
```

- 信号a 在硬件上是物理存在的。
- 信号a 和q 的值都被clk 上升沿激活而改变（两触发器级联）。


a
q~reg0
PRE
D
Q
ENA
CLR
d
clk
q


q~reg0
PRE
d
D
Q
q
clk
ENA
CLR
a
PRE
d
D
Q
q
clk
ENA
CLR
a
q~reg0
PRE
PRE
d
D
Q
D
Q
q
clk
ENA
ENA
CLR
CLR

# 内容提要

# VHDL 数据类型

**标量类型** Scalar 该类型的数值都是有具体大小的。
整数类型、实数类型、枚举类型和物理类型。

**复合类型** Composite 可能由相同类型元素（数组类型）或不同类型元素（记录类型）组成。
数组类型和记录类型。

**存取类型** Access 提供对给定类型的对象的存取（通过指针）。
存取类型等价于指针类型。

**文件类型** File 提供对包含一系列给定类型的数据对象的存取。
文件类型用于定义代表文件的对象。

**子类型** 可以通过预定义或用户定义的类型导出子类型。

# VHDL 数据类型

**标量类型** Scalar 该类型的数值都是有具体大小的。
整数类型、实数类型、枚举类型和物理类型。

**复合类型** Composite 可能由相同类型元素（数组类型）或不同类型元素（记录类型）组成。
数组类型和记录类型。

**存取类型** Access 提供对给定类型的对象的存取（通过指针）。
存取类型等价于指针类型。

**文件类型** File 提供对包含一系列给定类型的数据对象的存取。
文件类型用于定义代表文件的对象。

**子类型** 可以通过预定义或用户定义的类型导出子类型。

# 文字

- 一种具有“值”的符号，既不同于保留字，也不同于运算符。
- 6 类：整数、实数（浮点数）、字符、字符串、位串和物理量。

整数 实数 字符 字符串 位串 物理量

- 整数中不含小数点，而实数中含有小数点。
- 它们可以在任意两个相邻的数字之间插入下划线不影响数值大小。
- 123_456.999_999 ⇔ 123456.999999 不同于标识符。
- 字符文字用单引号括起来的单个 ASCII 字符，如'A'、'/' 等。
- 字符串文字用双引号括起来的一串 ASCII 字符，如"abc"、"1234" 等。
- 位串文字以进制声明符 B、O 或 X 为前导双引号括起来的数字序列，如B"1010_1010"、O"377"、X"5F5" 等。
- 物理文字由整数或实数文字与表示物理单位的标识符组成，如3.5ns、20pf 等。

# 文字

- 一种具有“值”的符号，既不同于保留字，也不同于运算符。
- 6 类：整数、实数（浮点数）、字符、字符串、位串和物理量。

**整数 实数 字符 字符串 位串 物理量**

- 整数中不含小数点，而实数中含有小数点。
- 它们可以在任意两个相邻的数字之间插入下划线不影响数值大小。
- 123_456.999_999 $\Leftrightarrow$ 123456.999999 不同于标识符。
- 字符文字用单引号括起来的单个 ASCII 字符，如'A'、'/' 等。
- 字符串文字用双引号括起来的一串 ASCII 字符，如"abc"、"1234" 等。
- 位串文字以进制声明符 B、O 或 X 为前导双引号括起来的数字序列，如B"1010_1010"、O"377"、X"5F5" 等。
- 物理文字由整数或实数文字与表示物理单位的标识符组成，如3.5ns、20pf 等。

# 文字

- 一种具有“值”的符号，既不同于保留字，也不同于运算符。
- 6 类：整数、实数（浮点数）、字符、字符串、位串和物理量。

**整数 实数 字符 字符串 位串 物理量**

- 整数中不含小数点，而实数中含有小数点。
- 它们可以在任意两个相邻的数字之间插入下划线不影响数值大小。
- 123_456.999_999 ⇔ 123456.999999 不同于标识符。
- 字符文字用单引号括起来的单个 ASCII 字符，如'A'、'/' 等。
- 字符串文字用双引号括起来的一串 ASCII 字符，如"abc"、"1234" 等。
- 位串文字以进制声明符 B、O 或 X 为前导双引号括起来的数字序列，如B"1010_1010"、O"377"、X"5F5" 等。
- 物理文字由整数或实数文字与表示物理单位的标识符组成，如3.5ns、20pf 等。

# ① 标量类型

标量类型是最基本的数据类型，包括整数、实数、枚举和物理类型 4 种。

```
1  TYPE address IS RANGE 15 DOWNTO 0;
2  TYPE analog IS RANGE 0.0 TO 5.0;
3  TYPE Bit IS ('0', '1');
4  TYPE Week IS (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
5  TYPE voltage IS RANGE 0 TO 10E12
6    UNITS
7      uv;
8      mv = 1000 uv;
9      v = 1000 mv;
10     kv = 1000 v;
11   END UNITS;
```

- STANDARD 标准包中预定义标量类型包括整数类型Integer、实数类型Real、枚举类型Bit Boolean Character Severity_Level 以及物理类型Time。

# 标准逻辑 *Std_Logic*

```
1 TYPE Std_ULogic IS ('U','X','0','1','Z','W','L','H','-');
```

- 'U' 未初始化值，系统电压建立（上电）时的初始值；
- 'X' 强未知值，强逻辑 0 遭遇强逻辑 1 的结果；
- '0' 强逻辑 0，简称逻辑 0；
- '1' 强逻辑 1，简称逻辑 1；
- 'Z' 高阻值，三态门处于高阻态；
- 'W' 弱未知值，弱逻辑 0 遭遇弱逻辑 1 的结果；
- 'L' 弱逻辑 0，下拉；
- 'H' 弱逻辑 1，上拉；
- '-' 无关，不可能值。

```
1 SUBTYPE Std_Logic IS Resolved Std_ULogic;
```

- 完整的概括了数字系统中所有可能的数据表现形式（代替Bit）。
- 通常只有'0'、'1'、'Z'和'-'可以被综合。

# 标准逻辑 *Std_Logic*

```
1 TYPE Std_ULogic IS ('U','X','0','1','Z','W','L','H','-');
```

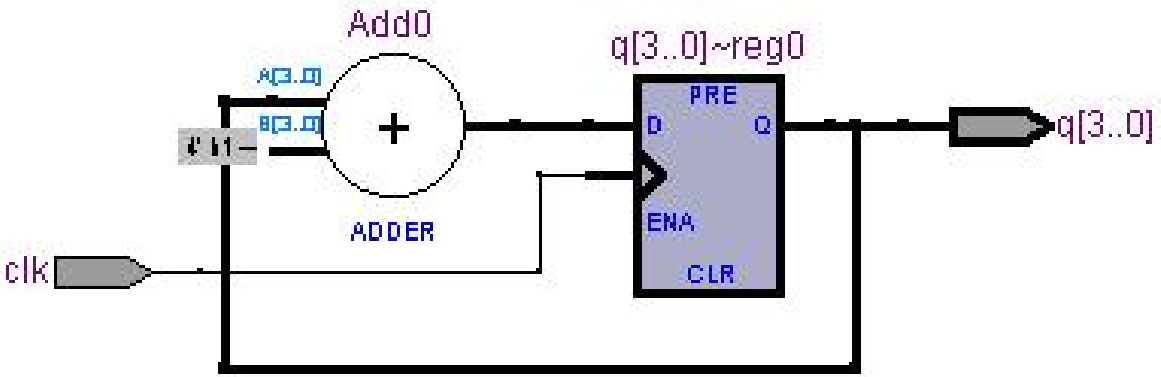
- 'U' 未初始化值，系统电压建立（上电）时的初始值；
- 'X' 强未知值，强逻辑 0 遭遇强逻辑 1 的结果；
- '0' 强逻辑 0，简称逻辑 0；
- '1' 强逻辑 1，简称逻辑 1；
- 'Z' 高阻值，三态门处于高阻态；
- 'W' 弱未知值，弱逻辑 0 遭遇弱逻辑 1 的结果；
- 'L' 弱逻辑 0，下拉；
- 'H' 弱逻辑 1，上拉；
- '-' 无关，不可能值。

```
1 SUBTYPE Std_Logic IS Resolved Std_ULogic;
```

- 完整的概括了数字系统中所有可能的数据表现形式（代替Bit）。
- 通常只有'0'、'1'、'Z' 和'-' 可以被综合。

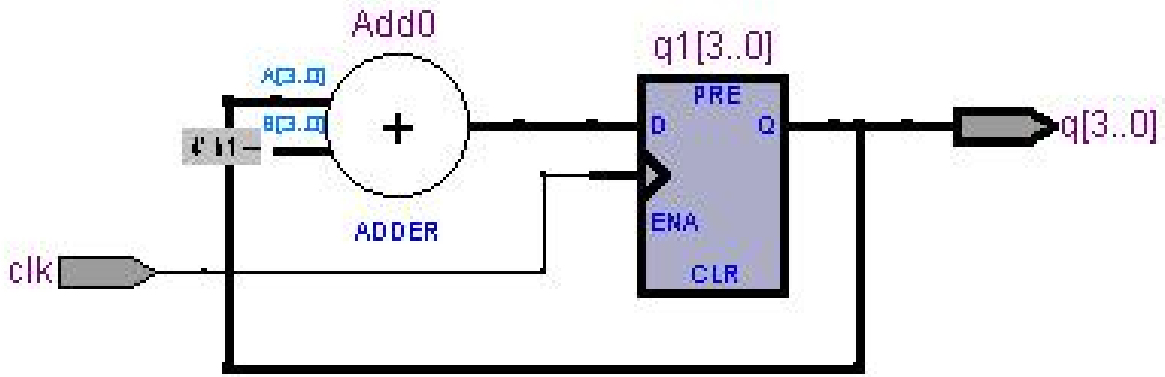
# 4位二进制加法计数器

```
1  LIBRARY IEEE;
2  USE IEEE.Std_Logic_1164.ALL;
3  USE IEEE.Std_Logic_Unsigned.ALL;

5  ENTITY cnt4 IS
6    PORT(clk: IN Std_Logic;q: BUFFER Std_Logic_Vector(3 DOWNTO 0));
7  END cnt4;
8  ARCHITECTURE bhv OF cnt4 IS
9  BEGIN
10   PROCESS(clk)
11   BEGIN
12     IF clk'Event AND clk='1' THEN
13       q <= q+1;
14     END IF;
15   END PROCESS;
16 END bhv;
```


Add0
A[3..0]
B[3..0]
VCC
ADDER
q[3..0]~reg0
PRE
D
Q
ENA
CLR
q[3..0]
clk

# 4位二进制加法计数器

```
1  LIBRARY IEEE;
2  USE IEEE.Std_Logic_1164.ALL;
3  USE IEEE.Std_Logic_Unsigned.ALL;

5  ENTITY cnt4 IS
6    PORT(clk: IN Std_Logic;q: OUT Std_Logic_Vector(3 DOWNTO 0));
7  END cnt4;
8  ARCHITECTURE bhv OF cnt4 IS
9    SIGNAL q1: Std_Logic_Vector(3 DOWNTO 0);
10 BEGIN
11   PROCESS(clk)
12   BEGIN
13     IF clk'Event AND clk='1' THEN
14       q1 <= q1+1;
15     END IF;
16   END PROCESS;
17   q <= q1;
18 END bhv;
```


Add0
A[3..0]
B[3..0]
VCC
ADDER
q1[3..0]
PRE
D
Q
ENA
CLR
q[3..0]
clk

# ② 复合类型

- 数组是同构复合类型，它所声明的数据是同一类型值的集合。
- 记录是异构复合类型，它所声明的数据可以是不同类型值的集合。

```
1 TYPE Byte IS ARRAY (7 DOWNTO 0) OF Bit;
2 TYPE Address IS RANGE 65535 DOWNTO 0;
3 TYPE Element IS ARRAY (Address) OF Byte;
4 TYPE Bit_Vector IS ARRAY (Natural RANGE <>) OF Bit;
5 TYPE Memory IS RECORD
6   Address: RANGE 16#FFFFF# DOWNTO 16#00000#;
7   Data: Byte;
8 END RECORD;
```

- array(7)
- record.Data

# 子类型

- 增加可重用性，而不声明太多的新类型。
- 子类型只对其基本类型（父类型）的值域加以限制，而不是一种新类型。
- 子类型和其父类型完全兼容，同一父类型的各子类型也相互兼容。
- 子类型的对象可以将其值直接赋给父类型的对象，而父类型对象在其值未超出子类型声明范围时也可以将其值直接赋给子类型对象；同一父类型各子类型之间同样。
- 建议不要创建过多的子类型。

```
1 TYPE Integer IS RANGE -2147483647 TO +2147483647;
2 SUBTYPE Natural IS Integer RANGE 0 TO Integer'High;
3 SUBTYPE Positive IS Integer RANGE 1 TO Integer'High;
```

# 类型转换

- VHDL 中每一个对象只能有一种类型。
- 如果赋值时对象的类型和值的类型不一致，则必须对该值的类型进行转换。
- 通常用函数或者常数来实现类型转换。

```
1 VARIABLE i: Integer;
2 VARIABLE r: Real;
3
4 r := Real(i);
5 i := Integer(r);
```

# 利用常数进行类型转换

```
1  LIBRARY IEEE;
2  USE IEEE.Std_Logic_1164.ALL;
3
4  ENTITY type_conv IS
5    PORT(s: IN Std_ULogic;b:OUT Bit);
6  END type_conv;
7  ARCHITECTURE arch OF type_conv IS
8    TYPE typeconv_typ IS ARRAY(Std_ULogic) OF Bit;
9    CONSTANT typeconv_con: typeconv_typ := ('0'|'L' => '0',
10                                         '1'|'H' => '1',
11                                         OTHERS => '0');
12 BEGIN
13   b <= typeconv_con(s);
14 END arch;
```

# 属性

- 属性是某一项目的特征，一个项目可以有多个属性。
- 如果某一项目的某个属性具有一个值的话，可以通过属性名来访问这个值。
- 可以具有属性的项目包括类型（子类型），信号、常量，实体、结构体、配置、程序包，过程、函数，元件，语句标号。
- 属性名的一般形式：`项目名'属性标识符`

```
1 Integer'High = 2147483647;
2 Integer'Low = -2147483647;
3 Natural'Left = 0;
4 Natural'Right = 2147483647;
5 Natural'Base'Left = -2147483647;
```

# 内容提要

# 运算符

1. ** ABS NOT
2. * / MOD REM
3. + - (Add Minus)
4. + - (Positive Negative)
5. SLL SLA SRL SRA ROL ROR
6. = /= < <= > >=
7. AND OR NAND NOR XOR XNOR

# ① 算术运算符

## 一元运算

- ABS + - (Positive Negative) 的操作数可以是任何数值类型（整数、实数和物理量）。

## 二元运算

- + - * / (Add Minus) 的操作数也可以是任何数值类型。
- MOD REM 运算的操作数只能是整数类型，结果也是整型。
- ** 运算的左操作数可以是整数或实数类型，但右操作数必须是整型，结果类型与左操作数相同。
- SLL SRL SLA SRA ROL ROR 左操作数必须是元素类型为Bit 或Boolean 的一维数组，右操作数必须是整型，结果的类型与左操作数类型相同。

# ① 算术运算符

## 一元运算

- ABS + - (Positive Negative) 的操作数可以是任何数值类型（整数、实数和物理量）。

## 二元运算

- + - * / (Add Minus) 的操作数也可以是任何数值类型。
  参加加、减运算的操作数类型必须相同，结果数值类型也相同。
  参加乘、除运算的操作数可以同为整数或者实数类型，结果数值类型不变。
  物理量可以被整数或者实数相乘或相除，结果仍未相同类型的物理量。
  物理量可以除以同一类型的物理量，但其结果为整型。
- MOD REM 运算的操作数只能是整数类型，结果也是整型。
- ** 运算的左操作数可以是整数或实数类型，但右操作数必须是整型，结果类型与左操作数相同。

# ① 算术运算符

## 一元运算

- ABS + - (Positive Negative) 的操作数可以是任何数值类型（整数、实数和物理量）。

## 二元运算

- + - * / (Add Minus) 的操作数也可以是任何数值类型。
- MOD REM 运算的操作数只能是整数类型，结果也是整型。
  取模的结果符合与右操作数相同，取余的结果符合与左操作数相同。
  如果要使结果被综合，参与取模和取余运算的操作数应是 2 的整数次幂。
- ** 运算的左操作数可以是整数或实数类型，但右操作数必须是整型，结果类型与左操作数相同。
- SLL SRL SLA SRA ROL ROR 左操作数必须是元素类型为Bit或Boolean 的一维数组，右操作数必须是整型，结果的类型与左操作数类型相同。

# ① 算术运算符

## 一元运算

- ABS + - (Positive Negative) 的操作数可以是任何数值类型（整数、实数和物理量）。

## 二元运算

- + - * / (Add Minus) 的操作数也可以是任何数值类型。
- MOD REM 运算的操作数只能是整数类型，结果也是整型。
- ** 运算的左操作数可以是整数或实数类型，但右操作数必须是整型，结果类型与左操作数相同。
  只有当左操作数为实数类型时，右操作数才可以为负整数。
- SLL SRL SLA SRA ROL ROR 左操作数必须是元素类型为Bit或Boolean 的一维数组，右操作数必须是整型，结果的类型与左操作数类型相同。

# ① 算术运算符

## 一元运算

- ABS + - (Positive Negative) 的操作数可以是任何数值类型（整数、实数和物理量）。

## 二元运算

- + - * / (Add Minus) 的操作数也可以是任何数值类型。
- MOD REM 运算的操作数只能是整数类型，结果也是整型。
- ** 运算的左操作数可以是整数或实数类型，但右操作数必须是整型，结果类型与左操作数相同。
- SLL SRL SLA SRA ROL ROR 左操作数必须是元素类型为Bit或Boolean 的一维数组，右操作数必须是整型，结果的类型与左操作数类型相同。

# ② 逻辑运算符

- 逻辑运算的操作数为Bit 或Boolean，结果类型不变。
- 操作数也可以是元素类型为Bit 或Boolean 的、同样长度的一维数组，运算被施加于每个数组的相应元素，结果为长度相同的数组。
- 无左右运算顺序，必须使用括号把多个逻辑运算符结合在一起。

一元运算 NOT

二元运算 AND OR XOR NAND NOR XNOR

# ② 逻辑运算符

- 逻辑运算的操作数为Bit 或Boolean，结果类型不变。
- 操作数也可以是元素类型为Bit 或Boolean 的、同样长度的一维数组，运算被施加于每个数组的相应元素，结果为长度相同的数组。
- 无左右运算顺序，必须使用括号把多个逻辑运算符结合在一起。

一元运算 NOT

二元运算 AND OR XOR NAND NOR XNOR

# ③ 关系运算符

**二元运算** = /= < <= > >=

- 左、右操作数类型必须相同，运算结果为Boolean 类型。
- = /= 操作数可以是除文件类型以外的任何数据类型。
- < <= > >= 操作数必须是标量类型，或者是元素类型为离散类型（整数类型或枚举类型）的一维数组。

# ④ 连接运算符

**二元运算** &

- 用于一维数组的连接。
- 将右操作数连接在左操作数后面，结果形成一个新的一维数组。
- 可以是两个一维数组相连（要求两数组类型相同），也可以是一个一维数组与一个新元素相连（要求新元素与数组元素类型相同），还可以两个元素连接形成数组（要求两元素类型相同）。

# 聚合赋值

- 当给一个数组中的元素赋予相同的值，特别是这个数组较大时，采用聚合赋值描述非常方便。
- 在聚合赋值中OTHERS 声明只能放在最后。

```
SIGNAL a: Std_Logic_Vector(7 DOWNTO 0);

a <= "01100000";
a <= (0, 1, 1, 0, 0, 0, 0, 0);
a <= '0'&'1'&'1'&'0'&'0'&'0'&'0'&'0';
a <= (6 => '1', 5 => '1', OTHERS => '0');
```

# 聚合赋值

- 当给一个数组中的元素赋予相同的值，特别是这个数组较大时，采用聚合赋值描述非常方便。
- 在聚合赋值中OTHERS 声明只能放在最后。

```
1 SIGNAL a: Std_Logic_Vector(7 DOWNTO 0);
2
3 a <= "01100000";
4 a <= (0, 1, 1, 0, 0, 0, 0, 0);
5 a <= '0'&'1'&'1'&'0'&'0'&'0'&'0'&'0';
6 a <= (6 => '1', 5 => '1', OTHERS => '0');
```