VHDL语言基础（二）

标识符

VHDL语言中符号书写的基本规则

VHDL’87版标识符的语法规则

- 短标识符

VHDL’93版标准

- 全部接受 VHDL’87版标识符的语法规则
- 扩展标识符

短标示符规则

有效字符为：英文字母（‘ a～z’、‘ A～Z’），数字（‘ 0～9’）和下划线（‘ _’）

第一个字符必须是字母。

下划线前后都必须有英文字母或数字。

最后一个字符不能是下划线。

不允许连续两个下划线。

VHDL的保留字不能用于标识符。

在标识符中大写字母和小写字母是等效的

```
_txclk    -- 标识符必须起始于字母        8B10B   -- 标识符必须起始于字母
large#number    -- 只能是数字、字母和下划线
link__bar     -- 不能有两个连续的下划线
rx_clk_     -- 最后字符不能是下划线
```

扩展标识符规则

- 扩展标识符用反斜杠来界定

  \control\、\score\都是合法的扩展标识符。

- 允许包含图形符号和空格符

  \mode of control\、\$100\都是合法的扩展标识符。

- 两个反斜杠之间的字可以和保留字相同

  \entity\、\select\都是合法的扩展标识符。

- 两个反斜杠之间的字可以用数字打头

  \80entity\、\5select\都是合法的扩展标识符。

- 扩展标识符中允许多个下划线相邻

  \80entity__adder\是合法的扩展标识符。

- 扩展标识符区分大小写

  \entity\和\ENTITY\是不同的标识符。

- 扩展标识符和短标识符不同

  \count\和Count、count是不同的标识符。

多选题 1分

设置

下列标识符哪些是合法的：

A 38DECODER

B ENCODER$

C MUX2_4_

D video_processor

提交

数据对象

凡是可以赋予一个值的客体就称为对象

数据对象

- 常数
- 信号
- 变量
- 文件

数据对象在使用前必须给予说明

常数

- 在设计描述中不会变化的值
- 就是对某一常数名赋予一个固定的值
- 在程序包、实体、结构体、进程和子程序的说明区域中说明
  - 其有效范围也相应限定
- 增强程序的可读性，便于修改程序

常数说明的一般格式

```
CONSTANT 常数名：数据类型 := 表达式;
```

下面的常数可以用来表示寄存器的宽度

```
CONSTANT WIDTH： INTEGER：= 8;
```

信号

- 是电子电路内部硬件连接的抽象，代表连线
- 在实体说明中，端口默认为信号
- 可以是逻辑门的输入或输出
- 也能表达存储元件的状态

信号说明的一般格式

SIGNAL 信号名：数据类型 约束条件 := 表达式；

SIGNAL sys_clk：BIT := ‘0’；

信号可以包含一个初始值，仿真时在开始设定的一个起始值

信号

- 信号可以在实体的说明部分、结构体的说明部分和程序包的说明部分声明
  - 其有效范围也相应限定

信号的赋值　“<=”符号

```
S1 <= S2 AFTER 10ns;
```

信号赋值时可以附加延时

变量

- 变量仅仅用于进程和子程序
- 必须在进程或子程序的说明区域加以说明
- 变量不能表达连线或存储单元
- 通常用于局部的数据存储，没有物理意义

变量说明的一般格式

VARIABLE 变量名：数据类型 约束条件 := 表达式;

VARIABLE COUNT : INTEGER RANGE 0 TO 255 : = 10;

变量用 := 进行赋值

变量可以包含一个初始值

信号与变量的区别

```
architecture rtl of start  is
signal count : integer range 0 to 7;
begin
   process(clk)
   begin
      if (clk'event and clk='1')
then
          count <= count + 1;
          if(count=0) then
              carryout <= '1';
          else
              carryout <= '0';
          end if;
      end if;
   end process;
end rtl;
```

```
architecture rtl of start  is
begin
process(clk)
variable count : integer range 0 to 7;
begin
      if (clk'event and clk='1') then
          count := count + 1;
          if(count=0) then
              carryout <= '1';
          else
              carryout <= '0';
          end if;
      end if;
   end process;
end rtl;
```

多选题 1分

设置

下列说法哪些是正确的

A 信号代表连线，由硬件对应物

B 变量用于局部数据存储，没有实际对应物

C 带延时的信号赋值语句可以实现延时器

D 给信号赋初值，当电路上电时就从这个初始状态开始工作

提交

数据类型

VHDL语言中的对象：信号、变量和常数都要指定数据类型

数据类型

数据类型

- 标准的数据类型
- 用户自定义数据类型

数据类型的定义严格

- 不同类型之间的数据不能直接代入
- 数据类型相同，位长不同也不能直接代入

标准的数据类型

1. 整数（INTEGER）

- 与数学中的整数的定义相同
  - 表示范围为 - 2147483647 ~ 2147483647
- 任何带有小数点的数字都被认为是实数
- 整数不能看作是位矢量
  - 不能按位来进行访问
  - 对整数不能用逻辑操作符

标准的数据类型

2. 位（BIT）

- 位值的表示方法是用 ‘0’或 ‘1’表示
- 可以用来描述数字系统中总线的值

3.位矢量（BIT_VECTOR）

- 是用双引号括起来的一组数据

  “001100”

- 用位矢量数据表示总线状态最形象也最方便

标准的数据类型

4.布尔量（BOOLEAN）

- 布尔量具有两种状态，“真”或者“假”
- 和位不同，没有数值的含义
  - 不能进行算术运算
  - 可以进行关系运算

标准的数据类型

5.大于等于零的整数（NATURAL）
正整数（POSITIVE）

- NATURAL类数据只能取值0和0以上的正整数
- POSITIVE则只能为正整数

用户定义的数据类型

1.枚举类型

- 枚举类型就是把类型中的各个元素都枚举、列表出来
- 枚举类型中的所有值都是用户定义的
  - 这些值可以是标识符，也可以是单个字符
- 典型的用法是用来定义状态机中的状态

枚举类型的定义格式

TYPE 数据类型名 IS（元素，元素，…）；

```
TYPE states IS (idle, ready, busy, error);
SIGNAL current_state: states;
```

定义的枚举类型可用于信号的说明

用户定义的数据类型

2.整数类型，实数类型

- 整数类型和实数类型在VHDL语言中已存在
- 是整数和实数类型的一个子类

整数或实数用户定义数据类型格式

```
TYPE  数据类型名  IS  数据类型定义  约束范围;
```

```
TYPE  digit  IS  INTEGER  RANGE  0  TO  9;
```

用户定义的数据类型

3.数组

- 数组是将相同类型的数据集合在一起形成的一个新的数据类型
  - 可以是一维的也可以是多维的
  - 通过数组下标访问数组中的任何一个元素

数组定义的格式

```
TYPE 数据类型名 IS ARRAY 范围 OF 原数据类型名;
```

```
TYPE word IS ARRAY (1 TO 8 ) OF STD_LOGIC;
SIGNAL MY_ARRAY : WORD;
        ……
MY_ARRAY(3) <= ‘0’;
        ……
```

用户定义的数据类型

4.记录类型

- 把多种不同类型的数据对象组织在一起
- 可以包含任意类型的数据对象，包括数组和记录
- 一个记录的各个字段可由元素名访问，从记录数据类型中提取元素数据类型时使用“.”

记录数据类型的定义格式

```
TYPE  数据类型名  IS  RECORD
        元素名：  数据类型名;
        元素名：  数据类型名;
        ……
END  RECORD [数据类型名];
```

```
TYPE IOCELL IS RECORD
        BUFFER_INP:  BIT_VECTOR (7 DOWNTO 0) ;
        ENABLE:  BIT;
        BUFFER_OUT:  BIT_VECTOR (7 DOWNTO 0) ;
END RECORD;
        ……
SIGNAL BUSA,  BUSB,  BUSC:  IOCELL;
SIGNAL VEC:  BIT_VECTOR (7 DOWNTO 0) ;
        ……
        BUSA.BUFFER_INP <= VEC;
        BUSB.BUFFER_INP <= BUSA.BUFFER_INP;
        BUSB.ENABLE <= '1';
        BUSC <= BUSB;
```

IEEE标准的“STD_LOGIC”和“STD_LOGIC_VECTOR”类型

- IEEE标准中定义了“STD_LOGIC”和“STD_LOGIC_VECTOR”型数据

使用时必须写出下列库说明语句和使用程序包说明语句

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

具有如下9种不同的值

‘U’, —— 未初始化（Uninitialized）
‘X’, —— 未知（Forcing Unknown）
‘0’, —— 强0（Forcing 0）
‘1’, —— 强1（Forcing 1）
‘Z’, —— 高阻（High Impedance）
‘W’, —— 弱未知（Weak Unknown）
‘L’, —— 弱0(Weak 0)
‘H’, —— 弱1（Weak 1）
‘-’, —— 不可能情况（Don’t care）

多选题 1分

设置

此题未设置答案，请点击右侧设置按钮

关于数据类型，下面说法正确的是：

A 整数可以综合为二进制数，因此在代码中可以按位访问

B bit类型可以表示‘0’、‘1’和高阻

C std_logic类型可以表示‘0’、‘1’和高阻

提交

运算操作符

运算操作符

- 逻辑运算
- 关系运算
- 算术运算
- 并置运算

操作数的类型应该和操作符所要求的类型相一致

运算操作符是有优先级

逻辑运算符

NOT ——取反；
AND ——与；
OR ——或；
NAND ——与非；
NOR ——或非；
XOR ——异或；

- 可以对“STD_LOGIC”、“BIT”和“STD_LOGIC_VECTOR”型数据进行逻辑运算
- NOT的优先级最高

算术运算符

+ ——加;

- ——减;

* ——乘;

/ ——除;

MOD ——取模;

REM ——取余;

+ ——正;

- ——负;

** ——指数;

ABS ——取绝对值;

关系运算符

= ——等于；
/= ——不等于；
< ——小于；
<= ——小于等于；
> —— 大于；
>= ——大于等于；

- 关系运算符的左右两边是运算操作数
- 左右两边操作数的类型必须相同，但位长度不一定相同
- 等号“=”和不等号“/=”可以适用于所有的数据类型

并置运算符

- 并置运算符“&”用于位的连接，形成位矢量
- 可以把两个位矢量连接起来形成更大的位矢量

```
SIGNAL DATA_A : STD_LOGIC_VECTOR(3 DOWNTO 0);
        .....
DATA_C <= D1 & D2 & D3 & D4;
DATA_D <= DATA_C & DATA_A;
```