VHDL语言基础

VHDL语言基础

- 简介
- **VHDL**程序基本结构
- 数据对象、数据类型、运算符和属性
- 进程
- 顺序语句
- 并行语句

硬件描述语言

- 集成电路的描述方式
  - 门级描述
    - 随着设计规模增大难以管理
    - 要求更高抽象层次描述方法
  - 图形和布尔方程式的描述方式
    - 费时、易出错，且在方程式中寻找错误很困难
    - 原理图的保持比较困难，需要一个文本来描述其设计构思和功能
    - 图形的输入环境往往也是专用
  - 硬件描述语言 **HDL**
    - **Hardware Description Language**

硬件描述语言

- 用于电子系统硬件描述的语言
  - 与高层次的软件设计语言类似
  - 不同点
    - 主要目的是用来编写硬件设计文件并建立硬件器件的仿真模型
    - 语意和语法的定义是为了能够描述硬件的行为
- 功能
  - 在希望的抽象层次上，可以对设计进行精确而简练的描述
  - 在不同层次上都易于形成用语言模拟和验证的设计描述
  - 在自动设计系统中作为设计输入
  - 易于设计的修改，易于把相应的修改并入设计文件中

VHDL语言

- **Very High Speed IC Hardware Description Language**
- 开发始于美国国防部1981年的超高速集成电路计划
  - 由Intermetrics公司、IBM公司和Texas Instrument公司承担开发
  - 目的是为了给出一种与工艺无关的、支持大规模系统设计的标准方法和手段
- 1987年VHDL被正式确定为IEEE-1076标准
  - 后来又作了若干修改，形成了VHDL'93标准
  - 美国政府选定VHDL作为联邦信息处理标准

VHDL的优点

- 描述能力强
  - 具有功能强大的语言结构
  - 用简洁明确的代码描述复杂的逻辑设计
  - 它支持多层次的设计描述
  - 支持设计库和可重用设计模块
- 设计和工艺、器件无关
  - 设计并不依赖于工艺和器件
  - 同一设计，可以用多种不同的方法来实现
  - 设计人员可以专注于设计

VHDL的优点

- 可移植、设计易于共享和复用
  - 用VHDL描述的设计可以被多种工具支持
    - 从一种仿真工具移植到另一种仿真工具
    - 从一种综合工具移植到另一种综合工具
- 效率高，成本低
  - 语言描述快捷、方便，大大提高了数字系统的设计速度
  - 和可编程逻辑器件结合，可以使产品以很快的速度面市
  - VHDL代码可以很容易地转为ASIC的设计

VHDL程序的基本结构

- VHDL把任意复杂度的电路模块看作一个单元
  - 一个单元又可以分为接口部分和设计描述部分
- 一个VHDL程序包括五个部分
  - 实体（Entity）
    - 描述设计的外部接口信号和该设计单元的公共信息
  - 结构体（Architecture）
    - 描述该设计单元的行为、数据的流程或结构
  - 库（Library）
    - 存放已编译的实体、结构体、程序包等，用户可以直接引用
  - 程序包（Package）
    - 存放设计可以共享的数据类型、常数和子程序等
  - 配置（Configuration）

VHDL 大小写不敏感

MUX.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MUX IS
PORT(
        A, B : IN STD_LOGIC;
        SEL : IN STD_LOGIC;
        C : OUT STD_LOGIC);
END MUX;
ARCHITECTURE MUX_EXAMPLE OF MUX IS
BEGIN
        PROCESS(SEL, A, B)
        BEGIN
                IF SEL = '1' THEN
                        C <= A;
                ELSE
                        C <= B;
                END IF;
        END PROCESS;
END MUX_EXAMPLE;
```

库

程序包

实体

结构体

文件名和实体名一致

每行；结尾

关键字end后跟实体名

关键字begin

关键字end后跟构造体名

实体

- **VHDL**中的基本单元和最重要的抽象
  - 可以代表整个系统
  - 可以代表一块电路板
  - 可以代表一个芯片
  - 可以代表一个单元或一个门电路
- 组成
  - 实体名
  - 类属表
  - 端口表
  - 实体说明部分
  - 实体语句
- 给实体命名，并给实体定义一个接口，接口信息用于和其他模块通信
  - 从某种程度上讲，实体是一个器件的外部视图

实体说明格式

```
ENTITY 实体名 IS
        [GENERIC （类属表）;]
        [PORT （端口表）；]
        [实体说明部分；]
        [BEGIN
                实体语句部分；]
END [ENTITY] [实体名]；
```

类属说明

- 为设计实体和其外部环境通信的静态信息提供通道
  - 用来规定端口的大小
  - 实体中的子元件数目
  - 实体的定时特性等
- 是可选项，放在端口说明之前
- 类属说明的一般格式

GENERIC ([COSTANT] 名字表：**[IN]** 子类型标示**[:=** 静态表达式**], ……);**

例如：

GENERIC(N：POSITIVE:= 3);

端口说明

- 每一个I/O信号都被称为端口，其功能对应于电路图符号的一个引脚
- 每个端口必须有
  - 一个名字
  - 通信模式
    - 说明数据通过该端口的流动方向
  - 数据类型
    - 说明流过该端口的数据类型
- 端口说明的一般格式

PORT（[SIGNAL] 名字：[模式] 子类型标识 [BUS] [：= 静态表达式]，….）；

通信模式

- **IN模式**
  - 输入模式，只允许信号流入实体
- **OUT模式**
  - 输出模式，只允许信号流出实体
- **INOUT模式**
  - 双向模式，既可以流入，也可以流出
  - 双向模式可以替代输入、输出和缓冲模式
  - 实际的设计中，只有纯粹的双向信号才使用双向模式
- **BUFFER模式**
  - 缓冲模式，和输出模式的端口类似
  - 既可以用于输出，也可以用于反馈

out与buffer的区别

```
entity test1 is
   port(a: in std_logic;
      b,c: out std_logic );
end test1;

architecture a of test1 is
begin
      b <= not(a);
      c <= b;--Error
end a;
```

```
entity test2 is
     port(a: in std_logic;
       b: buffer std_logic;
       c: out std_logic );
end test2;

architecture a of test2 is
begin
        b <= not(a);
        c <= b;
 end a;
```

数据类型

- **IEEE1076/93标准规定的数据类型**
  - 布尔型（**boolean**）
  - 位型（**bit**）
  - 位矢量型（**bit_vector**）
  - 整数型（**integer**）

```
ENTITY  MUX  IS
      GENERIC（DELAY : =  5ns）;
      PORT（
            IN1, IN2, SEL  : IN  BIT;
            OUTPUT :  OUT  BIT）;
END  ENTITY  MUX ;
```

实体说明部分和语句部分

- 说明部分是这个实体中的公共信息
  - 可以声明和定义子程序、类型、信号、常量等，也可以使用**USE**子句
- 实体语句部分可以包含一系列语句
  - 并不打算表现元件的动态特性，而只是对元件接口进行检验和判断

结构体

- 描述实体的内在
  - 描述一个实体的功能
  - 规定实体的数据流程
  - 定义实体中内部单元的连接关系
- 结构体的一般格式

```
ARCHITECTURE 结构体名 OF 实体名 IS
        [说明部分]
BEGIN
        [并行语句]
END 结构体名;
```

对结构体内部所使用的信号、常数、数据类型、元件和子程序等进行声明和定义

用于具体地描述结构体的行为及其连接关系

行为描述

- 对设计中的描述是按算法的路径来描述的
- 优点在于无需关注设计的门级实现，而只需注意正确的函数模型

例：等值比较器的功能，当信号**A**和**B**相等时，**EQUALS**有效

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
ENTITY  EQCOMP4  IS
       PORT(
              A, B : IN STD_LOGIC_VECTOR(3  DOWNTO 0);
              EQUALS:  OUT  STD_LOGIC);
END  EQCOMP4;

ARCHITECTURE  BEHAVAL  OF  EQCOMP4  IS
BEGIN
       COMP:  PROCESS(A, B)
       BEGIN
              IF  A = B  THEN
                     EQUALS <= '1';
              ELSE
                     EQUALS <= '0';
              END  IF;
       END  PROCESS  COMP;
END  BEHAVAL;
```

不考虑在电路中到底是怎样实现的。

数据流描述

- 描述数据流的运动路径和运动方向
- 主要使用并行信号赋值语句，既显式表示了该设计单元的行为，也隐式表示了该设计单元的结构

例：等值比较器的功能，当信号**A**和**B**相等时，**EQUALS**有效

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
ENTITY       EQCOMP4  IS
      PORT(
            A,  B: IN  STD_LOGIC_VECTOR(3  DOWNTO  0);
            EQUALS:  OUT  STD_LOGIC);
END  EQCOMP4;

ARCHITECTURE  DATAFLOW  OF  EQCOMP4  IS
BEGIN
      EQUALS <= '1' WHEN  (A = B) ELSE  '0';
END  DATAFLOW;
```

结构化描述

- 用VHDL来描述网表组成
- 和图形网表非常相似：元件被例化，且用信号互相连接
- 通常用于层次设计

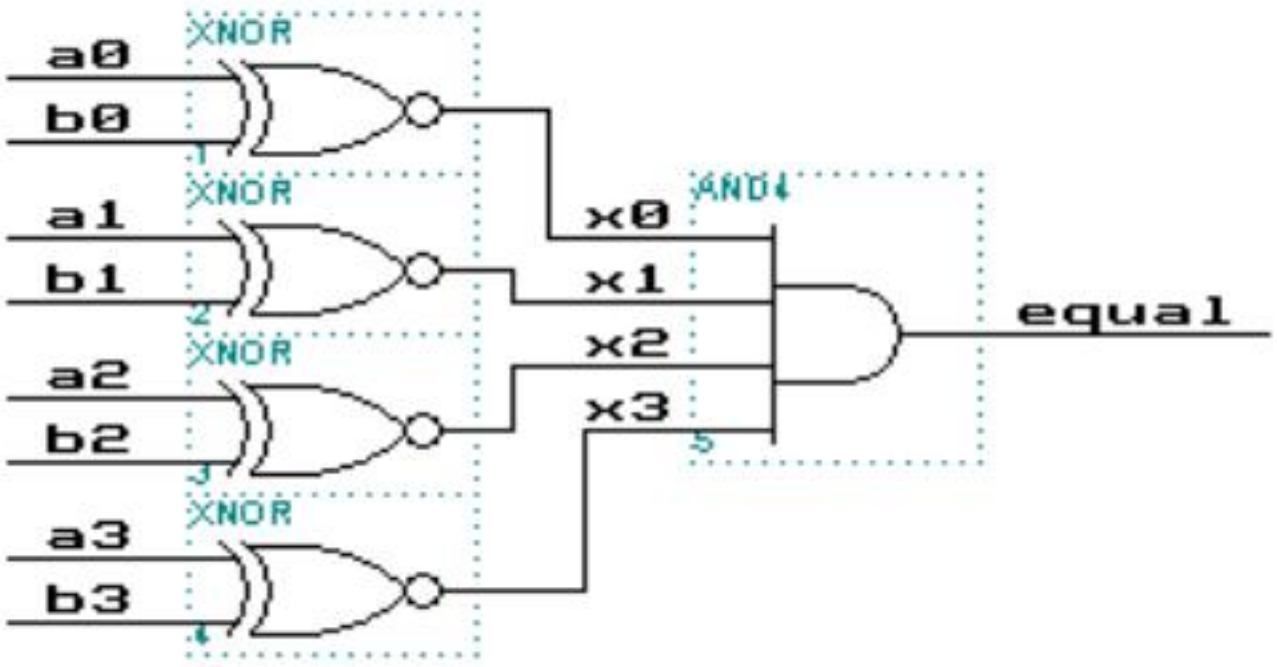
例：等值比较器的功能，当信号A和B相等时，EQUALS有效

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
USE  WORK.GATESPKG.ALL;
ENTITY        EQCOMP4  IS
      PORT(
            A,  B:IN  STD_LOGIC_VECTOR(3  DOWNTO  0);
            EQUALS:  OUT  STD_LOGIC);
END  EQCOMP4;
```

```
ARCHITECTURE  STRUCTURE  OF  EQCOMP4  IS
        COMPONENT XNOR2
        PORT(
                A, B : IN STD_LOGIC;
                C : OUT STD_LOGIC);
        END COMPONENT;
        COMPONENT AND4
        PORT(
                A, B, C, D : IN STD_LOGIC;
                E : OUT STD_LOGIC);
        END COMPONENT;
        SIGNAL  X : STD_LOGIC_VECTOR (0 TO 3) ;
BEGIN
        U0: XNOR2  PORT  MAP (A (0) , B (0) , X (0) ) ;
        U1: XNOR2  PORT  MAP (A (1) , B (1) , X (1) ) ;
        U2: XNOR2  PORT  MAP (A (2) , B (2) , X (2) ) ;
        U3: XNOR2  PORT  MAP (A (3) , B (3) , X (3) ) ;
        U4: AND4  PORT  MAP (X (0) , X (1) , X (2) , X (3) ,
                                        EQUALS) ;
END  STRUCTURE;
```

•类似于电路的网络表，将各个器件通过语言的形式进行连接，与电路有一一对应的关系。

•一般用于大规模电路的层次化设计时。



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。

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

```
LIBRARY IEEE;

USE IEEE.std_logic_1164.ALL;

PACKAGE sigdecl_pkg IS

TYPE bus_type IS ARRAY(0 to 7) OF std_logic;

SIGNAL vcc : std_logic := '1';

SIGNAL ground : std_logic := '0';

FUNCTION magic_function(a : IN bus_type)

RETURN bus_type;

END sigdecl_pkg;
```

```
USE WORK.sigdecl_pkg.ALL;

LIBRARY IEEE;

USE IEEE.std_logic_1164.ALL;

ENTITY sig_test is

PORT(

     data_in : IN bus_type;

     data_out : OUT bus_type);

     SIGNAL sys_clk : std_logic := '1';

END sig_test;
```

```
ARCHITECTURE data_flow OF sig_test IS
     SIGNAL int_bus : bus_type;
     CONSTANT disconnect_value : bus_type :=
          ('X','X','X','X','X','X','X','X');
BEGIN
   int_bus <= data_in WHEN sys_clk = '1'
                  ELSE int_bus;
   data_out <= magic_function(int_bus) WHEN
            sys_clk = '0'ELSE disconnect_value;
    sys_clk <= NOT(sys_clk) after 50 ns;
END data_flow;
```

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;
```

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
  - 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

2.实数（REAL）

- 实数值的范围为－1.0E＋38～＋1.0E＋38
- 实数有正负，书写时应有小数点

标准的数据类型

3. 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

2.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

5.字符（CHARACTER）

- 定义的字符量通常用单引号括起来，如‘A’
- 字符量中的大小写字符则认为是不一样的
- 字符包括A~Z、a～z、0～9、空格及一些特殊字符

标准的数据类型

6.字符串（STRING）

- 由双引号括起来的一个字符序列，也称字符矢量或字符串数组，如“integer range”
- 常用于程序的提示和说明

7.时间（TIME）

- 时间是一个物理数据类型
- 完整的时间量数据应包含整数和单位两部分，整数和单位之间至少应留一个空格的位置。例如，55 sec，2 min
- 包集合STANDARD中给出了时间的预定义，其单位为fs，ps，ns，us，ms，sec，min，hr
- 时间类型一般用于仿真，而不用于逻辑综合

标准的数据类型

8.错误等级（SEVERITY LEVEL）

- 用来表示系统的状态
  - **NOTE**（注意）
  - **WARNING**（警告）
  - **ERROR**（出错）
  - **FAILURE**（失败）
- 在系统仿真过程中可以用这四种状态来提示系统的当前工作状态

标准的数据类型

9.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）

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;
```

属性的描述

- 预定义属性函数
  - 数值类属性函数
  - 函数类属性函数
  - 信号类属性函数
  - 数据类型类属性函数
  - 数据范围类属性函数
- 通过属性描述语句，可以得到对象的有关值、功能、类型和范围（区间）的信息

数值类属性

1.一般数据的数值属性

书写格式

对象’属性名

- 左边界值（LEFT）
- 右边界值（RIGHT）
- 上限值（HIGH）
- 下限值（LOW）

TYPE word IS ARRAY（31 DOWNTO 0）OF BIT；

word’LEFT —— word的左边界值31

word’RIGHT —— word的右边界值0

word’HIGH —— word的上限值31

word’LOW —— word的下限值0

数值类属性

2.数组的数值属性

- **LENGTH**
- 用该属性得到一个数组的长度值

```
TYPE  PCBUS  IS  ARRAY （0 TO 15）OF  BIT

L :=  PCBUS’LENGTH
```

可以获得数组**PCBUS**的长度

函数类属性

1.数据类型属性函数

- 可以得到有关数据类型的各种信息

T'POS（x） ——得到输入**x**值的位置序号

T'VAL（x） ——得到输入位置序号**x**的值

T'SUCC（x） ——得到输入**x**值的下一个值

T'PRED（x） ——得到输入**x**值的前一个值

T'LEFTOF（x） ——得到邻接输入**x**值左边的值

T'REGHTOF（x） ——得到邻接输入**x**值右边值

TYPE time1 IS（sec，min，hour，day，month，year）；

time1’POS（hour）　——元素hour的位置序号2

time1’VAL（3）　——位置序号为3的元素值day

time1’SUCC（hour）　——hour的下一个值day

time1’PRED（hour）　——hour的前一个值min

time1’LEFTOF（hour）　——hour左边的邻值min

time1’REGHTOF（hour）——hour右边的邻值day

函数类属性

2.数组属性函数

- 可以得到可得到数组的区间

T'LEFT（n） ——得到索引号为n的区间的左边界值。在这里n实际上是多维数组中所定义的多维区间的序号。当n缺省时，就代表对一维区间进行操作。

T'RIGHT（n） ——得到索引号为n的区间的右边界值

T'HIGH（n） ——得到索引号为n的区间的高端的上限值

T'LOW（n） ——得到索引号为n的区间的低端的下限值

信号属性函数

- 用来得到信号的行为信息和功能信息
  - 信号的值是否有变化
  - 最后一次变化之前的值为多少
  - 从最后一次变化到现在经历了多长时间
- 常用属性有
  - **EVENT**
  - **LAST_VALUE**
  - **LAST_EVENT**

signal'event ——如果在当前相当小的一段时间间隔内，该事件发生，返回一个布尔量为“真”，否则返回“假”

signal'active ——在当前时间内，如果信号发生了变化，事件作了处理，则返回“真”，否则返回了“假”

signal'last_event ——信号最后一次改变到现在时刻所经历的时间，并将这段时间值返回

signal'last_value ——信号最后一次变化前的值，并将这个历史值返回

signal'last_active ——该属性函数返回一个时间值，即从信号前一次改变到现在的时间长度

VHDL语言的基本语法

- VHDL语言提供了一系列顺序语句和并行语句
  - 顺序语句的用来实现模型的算法部分
  - 并行语句用来表示黑盒子的连接关系
- 结构体是由一系列的并行语句构成
  - 每个并行语句表示一个功能单元
  - 多个功能单元构成一个结构体

ARCHITECTURE 结构体名 OF 实体名 IS

——说明部分

BEGIN

——并行语句A

——并行语句B

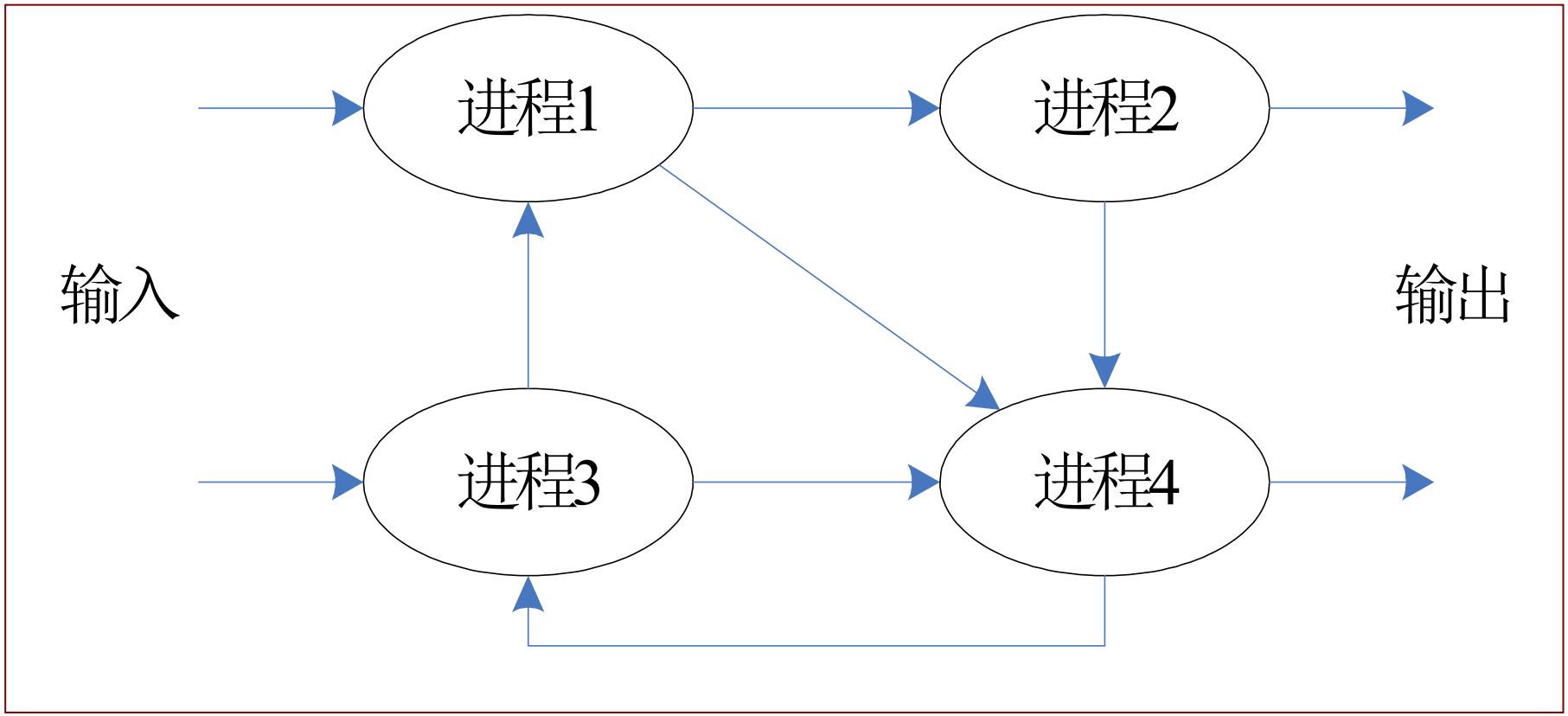
——并行语句C

END 结构体名；

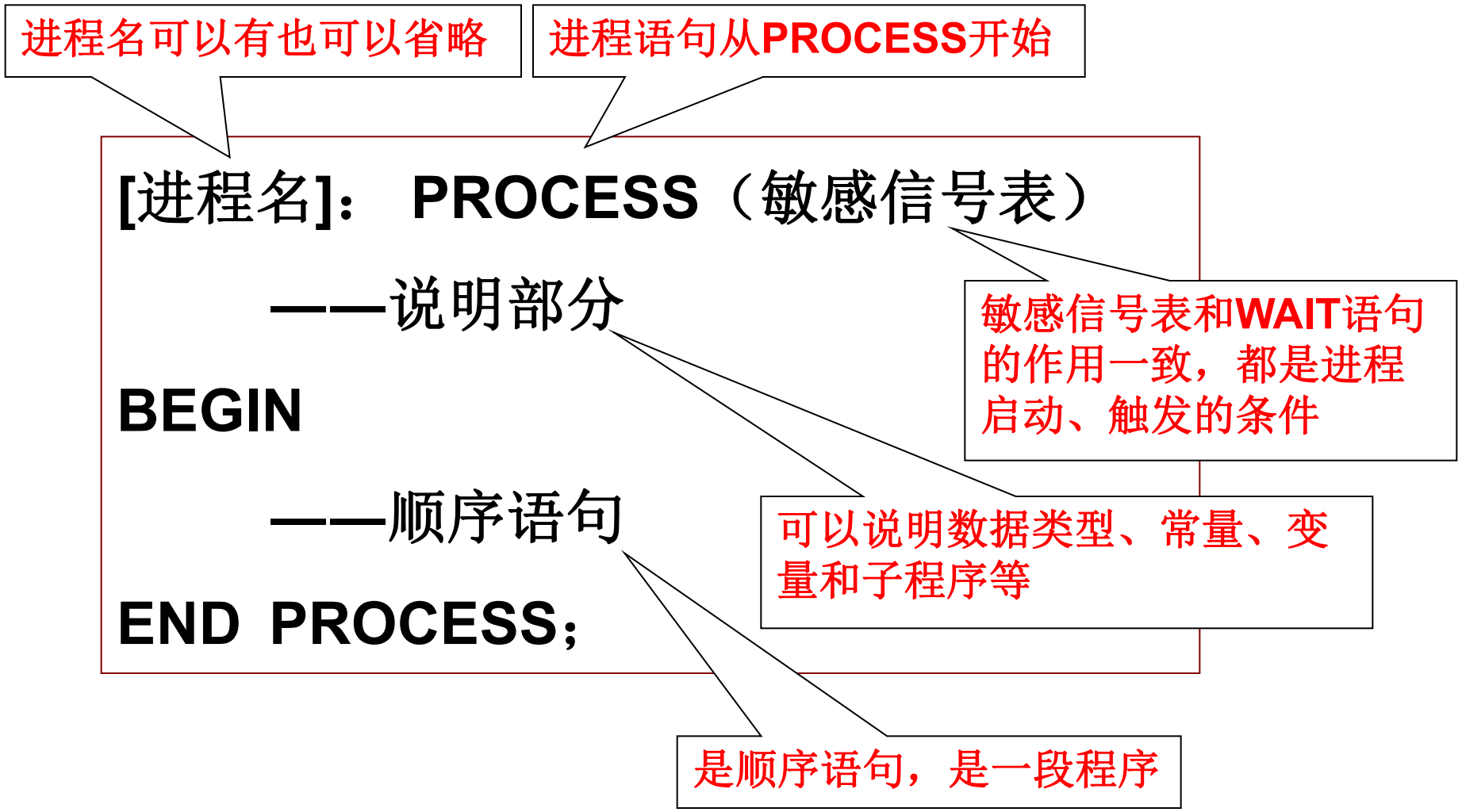
VHDL模型的最基本的表示方法

- 并行执行的进程（PROCESS）
  - 进程语句是并行语句，进程语句之间是并行关系
  - 进程语句内部则是由一组在整个模拟期间连续执行的顺序语句构成，是顺序执行的
  - 进程之间通过信号或共享变量进行通信
- PROCESS是描述硬件并行工作的最常用、最基本的语句

进程模型图



进程语句（PROCESS）



```
ENTITY  DFF  IS
        PORT（
                D，CLK：  IN  BIT；
                Q：  OUT  BIT）；
END DFF；
ARCHITECTURE    BEHAVE  OF  DFF  IS
BEGIN
        P1：PROCESS（CLK）
        BEGIN
                IF  CLK’EVENT AND CLK = ‘1’  THEN
                        Q <= D  AFTER  10  ns；
                END  IF；
        END  PROCESS  P1；
END  BEHAVE；
```

- **PROCESS**语句中含有敏感表，则等价于该进程语句内的最后一个语句是一个隐含的**WAIT**语句
- 含有敏感信号表的进程语句中不允许再显式出现**WAIT**语句
- 不含有敏感信号表的进程中可以有多个显式的**WAIT**语句
  - **WAIT**语句的执行会暂停进程的执行，直到敏感信号发生变化或某种条件满足为止

WAIT语句的格式

WAIT [ON 信号表] [UNTIL 条件] [FOR 时间表达式];

WAIT ——无限等待

WAIT ON ——等待敏感信号发生变化

WAIT UNTIL 表达式——等待表达式成立

WAIT FOR 时间表达式——等待一段由时间表达式指定的时间

```
PROCESS
        VARIABLE   TEMP：   BIT；
BEGIN
        TEMP  ：= A  OR  B；
        C <=   NOT   TEMP；
        WAIT   ON   A，   B；          -- 等待信号A或B的值改变
END   PROCESS；


WAIT  ON  A  FOR  50  ns;          -- 等待信号A的值改变或已过50ns的时间
WAIT  UNTIL  A = ‘0’;              -- 等待信号A的值为 ‘0’；
WAIT；                             -- 永远等待
```

进程语句的特点

- 一个结构体中可以有多个进程存在
- 进程之间并行运行，并可存取结构体或实体中所定义的信号；
- 进程内部的所有语句都是按顺序执行的；
- 为启动进程，在进程结构中必须包含一个显式的敏感信号量表或者一个**WAIT**语句；
- 进程之间的通信是通过信号传递来实现的

顺序语句

- 进程、过程和函数内部是顺序语句
  - 完全按程序中出现的顺序执行各条语句
  - 前面语句的执行结果可能直接影响后面语句的执行

- **WAIT**语句
- 变量赋值语句
- 信号赋值语句
- **IF**语句
- **CASE**语句
- **LOOP**语句
- **NEXT**语句
- **EXIT**语句
- **RETURN**语句
- **NULL**语句
- **REPORT**语句

顺序语句

1.变量赋值语句

- 变量的说明和赋值限定在顺序区域内
  - 只能在进程、函数和过程中
  - 信号值可以赋给变量，变量的值也可以赋给信号
- 变量的赋值是立即执行的

变量赋值语句的格式

目的变量 := 表达式；

```
SIGNAL   SIG ：  BIT：= ‘0’;
        ……
PROCESS
        VARIABLE   EVENT_ON_SIG ： INTEGER ： = 0;
BEGIN
        WAIT   ON   SIG;
        EVENT_ON_SIG ： = EVENT_ON_SIG + 1;
END   PROCESS;
```

变量值只是在进程或子程序中使用，无法传递到进程之外

顺序语句

2.信号赋值语句

- 是VHDL语言中进行行为描述的最基本的语句
- 信号的更新是在所有进程被执行并挂起后

变量赋值语句的格式

目的信号量 <= 信号量表达式

A <= B；

A得到B的值

A <= B AFTER 5 ns；

当B发生变化5ns以后才被代入到信号A

顺序语句

3. IF语句

- 根据所指定的条件来确定执行哪些语句
- 书写格式通常可以分为三种类型

格式 1

```
IF   条件   THEN
     ——顺序语句
END  IF;
```

- 如果条件成立，则执行IF语句所包含的顺序处理语句
- 如果条件不成立，程序将跳过IF语句所包含的顺序处理语句，而向下执行IF语句后的语句
- 这种描述可以用来描述D触发器

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY DFF IS
PORT（
        CLK，D： IN STD_LOGIC;
        Q: OUT STD_LOGIC）;
END DFF;
ARCHITECTURE BEHAVE OF DFF IS
BEGIN
        PROCESS（CLK）
        BEGIN
                IF（CLK'EVENT AND CLK = '1'）THEN
                        Q <= D;
                END IF;
        END PROCESS;
END BEHAVE;
```

格式 2

```
IF   条件   THEN
     ——顺序语句
ELSE
     ——顺序语句
END   IF;
```

- 指定的条件满足时，将执行**THEN**和**ELSE**之间所界定的顺序处理语句
- 当**IF**语句所指定的条件不满足时，将执行**ELSE**和**END IF**之间的顺序处理语句
- 用条件来选择两条不同程序执行的路径
- 描述的典型电路是二选一电路

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MUX2 IS
PORT（
        A，B，SEL：IN STD_LOGIC;
        C：OUT STD_LOGIC）;
END MUX2;
ARCHITECTURE BEHAVE OF MUX2 IS
BEGIN
        PROCESS（A，B，SEL）
        BEGIN
                IF（SEL = '1'）THEN
                        C <= A;
                ELSE
                        C <= B;
                END IF;
        END PROCESS;
END BEHAVE;
```

格式 3

```
IF   条件   THEN
     ——顺序语句
ELSIF   条件   THEN
     ——顺序语句
          ……
ELSIF   条件   THEN
     ——顺序语句
ELSE
     ——顺序语句
END   IF；
```

- 设置了多个条件，当满足所设的多个条件之一时，就执行该条件后跟的顺序处理语句
- 如果所有设置都不满足时，则执行**ELSE**和**END IF**之间的顺序处理语句
- 描述的典型电路是多选一电路

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY MUX4 IS
PORT（
        INPUT： IN STD_LOGIC_VECTOR（3 DOWNTO 0）;
        SEL： IN STD_LOGIC_VECTOR（1 DOWNTO 0）;
        C： OUT STD_LOGIC）;
END MUX4;
ARCHITECTURE BEHAVE OF MUX4 IS
BEGIN
        PROCESS（INPUT， SEL）
        BEGIN
                IF（SEL = “00”）THEN
                        C <= INPUT（0）;
                ELSIF（SEL = “01”）THEN
                        C <= INPUT（1）;
                ELSIF（SEL = “10”）THEN
                        C <= INPUT（2）;
                ELSE
                        C <= INPUT（3）;
                END IF;
        END PROCESS;
END BEHAVE;
```

顺序语句

4. CASE语句

- 从许多不同语句的序列中选择其中之一执行
- 常用来描述总线或编码译码的行为

CASE语句的格式

```
CASE   表达式   IS
    WHEN   条件表达式  =>
                        顺序语句;
END   CASE;
```

- 当CASE和IS之间的表达式的取值满足指定的条件表达式的值时，执行后跟的由符号 => 所指的顺序处理语句

```
ARCHITECTURE BEHAVE OF MUX4 IS
        SIGNAL SEL: INTEGER;
BEGIN
        PROCESS（A， B， I0， I1， I2， I3）
        BEGIN
                SEL <= 0;
                IF（A = ‘1’）THEN
                        SEL <= SEL + 1;
                END IF;
                IF（B = ‘1’）THEN
                        SEL <= SEL + 2;
                END IF;
                CASE SEL IS
                        WHEN 0 => Q <= I0;
                        WHEN 1 => Q <= I1;
                        WHEN 2 => Q <= I2;
                        WHEN 3 => Q <= I3;
                END CASE;
        END PROCESS;
END BEHAVE;
```

四选一选择器

顺序语句

5. LOOP语句

- **LOOP语句可以使程序能进行有规则的循环**
- **重复模式有两种：WHILE和FOR**

FOR循环

[标号]：FOR　循环变量　IN　离散范围　LOOP
——顺序语句；
END　LOOP　[标号]；

- **循环变量的值在每次循环中都将发生变化**
- **IN后跟的离散范围表示循环变量在循环过程中依次取值的范围**

I是循环变量，它可取值1，2，…，9共9个值

```
ASUM：  FOR  I  IN  1  TO  9  LOOP
        SUM  =  I  +  SUM;
END  LOOP  ASUM;
```

实现1～9的累加运算

注：

- 循环变量在信号说明和变量说明中都不能出现
- 信号和变量也不能代入到循环变量中

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
ENTITY  PARITY  IS
PORT（
        INPUT ：IN  STD_LOGIC_VECTOR(7 DOWNTO 0)；
        OUTPUT ：OUT  STD_LOGIC）；
END  PARITY;
ARCHITECTURE  BEHAVE OF PARITY IS
BEGIN
        PROCESS(INPUT)
                VARIABLE TEMP : STD_LOGIC;
        BEGIN
                TEMP := ‘0’;
                FOR I IN 0 TO 7 LOOP
                        TEMP := TEMP XOR INPUT(I);
                END LOOP;
                OUTPUT <= TEMP;
        END PROCESS;
END BEHAVE;
```

8位奇偶校验电路

WHILE循环

```
[标号]： WHILE 条件 LOOP
        ——顺序语句；
END LOOP [标号];
```

- 如果条件为“真”，则进行循环，如果条件为“假”，则结束循环

```
I ：= 1;
SUM ：= 0;
ABC： WHILE（I < 1 0 ）LOOP
        SUM ：= I + SUM;
        I ：= I + 1;
END LOOP ABC;
```

循环控制变量I的递增是通过算式I：=I+1实现的

顺序语句

6. NEXT语句

•在LOOP语句中NEXT语句用来跳出本次循环

格式

NEXT　[标号]　[WHEN条件]；

•NEXT语句执行时将停止本次迭代，而进入到下一次新的迭代

•NEXT后跟的标号表明下一次迭代的起始位置，WHEN条件表明NEXT语句执行的条件

•如果NEXT语句既无标号也无WHEN条件说明，那么只要执行到该语句，就立即无条件地跳出本次循环，从LOOP语句的起始位置进入下一次循环

```
FOR I IN 0 TO MAX LOOP
        IF（DONE（I） = TRUE）THEN
                NEXT;
        ELSE
                DONE（I） : = TRUE;
        END IF;
        Q（I） <= A（I） AND B（I）;
END LOOP;
```

顺序语句

7. EXIT语句

- LOOP语句中使用的循环控制语句
- 执行EXIT语句将结束循环状态，从LOOP语句中跳出，结束LOOP语句的正常执行

格式

```
EXIT  [标号]  [WHEN条件];
```

- 如果EXIT语句含有条件，条件为真时，从LOOP语句中跳出，条件为假时，则继续LOOP循环
- 如果EXIT语句含有标号，则跳到标号处继续执行

```
L1：  FOR  I  IN  10  DOWNTO  1  LOOP
        L2：  FOR  J  IN  0  TO  I  LOOP
                EXIT  L2  WHEN  I = J；
                MATRIX（I，J）：= I *（J + 1）；
        END  LOOP  L2；
END  LOOP  L1；
```

顺序语句

8. RETURN语句

- 是一段子程序结束后，返回主程序的控制语句
- 用来结束函数和过程的执行

格式

RETURN [表达式];

- 过程中的**RETURN**语句必须是无条件的，不能有表达式
- 函数语句中的**RETURN**语句必须有表达式 ,函数的结束必须使用**RETURN**语句

顺序语句

9. NULL语句

•表示无任何动作，执行NULL语句只是使程序走到下一个语句

格式

```
NULL;
```

•经常用在CASE语句中，用来表示在某种情况下不需要做任何动作

```
case controller_command is
        when forward => engage_motor_forward;
        when reverse => engage_motor_reverse;
        when idle => null;
end case;
```

并行语句

- 进程语句（**PROCESS**）
- 信号赋值语句（**SIGNAL ASSIGNMENT**）
- 块语句（**BLOCK**）
- 元件调用语句（**COMPONENT**）
- 端口映射语句（**PORT MAP**）
- 生成语句（**GENERATE**）
- 过程调用语句（**PROCEDURE CALL**）

并行语句

1.并行信号赋值语句

- 一个并行信号赋值语句代表着对该信号赋值的等价的进程语句

普通并行信号赋值语句

条件信号赋值语句

选择信号赋值语句

普通并行信号赋值语句

信号量 <= 敏感信号量表达式；

- 是VHDL语言最基本的语句之一
- 用在进程内部时，是作为顺序语句出现的
- 用在结构体中，进程的外部时，是作为并行语句出现的

```
Q <= A AND B AND C;
```

并行信号赋值语句
等效为一个进程

```
PROCESS(A, B, C)
BEGIN
        Q <= A AND B AND C;
END PROCESS;
```

条件信号赋值语句

```
目的信号量 <= 表达式1 WHEN 条件1 ELSE
              表达式2 WHEN 条件2 ELSE
              表达式3 WHEN 条件3 ELSE
                ……
              ELSE 表达式n;
```

- 每个表达式后面都跟有用**WHEN**所指定的条件，如果满足该条件，则该表达式值代入目的信号量
- 如果不满足条件，则再判断下一个表达式所指定的条件
- 最后一个表达式可以没有条件，它表明，在上述表达式所指明的条件都不满足时，则将该表达式的值代入目标信号量

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
ENTITY  MUX4  IS
PORT（
        I0，I1，I2，I3：  IN  STD_LOGIC;
        A，  B：  IN  STD_LOGIC;
        Q：  OUT  STD_LOGIC）;
END  MUX4;
ARCHITECTURE  RT  OF  MUX4  IS
        SIGNAL  SL：  STD_LOGIC_VECTOR（1 DOWNTO 0）;
BEGIN
        SEL <= B&A;
        Q <= I0  WHEN  SEL=“00”  ELSE
             I1  WHEN  SEL=“01”  ELSE
             I2  WHEN  SEL=“10”  ELSE
             I3  WHEN  SEL=“11”  ELSE
            ‘X’;
END  RT;
```

可以等效为用IF语句描述的进程

```
ARCHITECTURE  RT  OF  MUX4  IS
        SIGNAL  SL：  STD_LOGIC_VECTOR（1  DOWNTO  0）；
BEGIN
        SEL <=  B&A；
        PROCESS(SEL, I0, I1, I2, I3)
        BEGIN
                IF SEL=”00” THEN
                        Q <= I0;
                ELSIF SEL=”01” THEN
                        Q <= I1;
                ELSIF SEL=”10” THEN
                        Q <= I2;
                ELSIF SEL=”11” THEN
                        Q <= I3;
                ELSE
                        Q <= ‘X’;
                END IF;
        END PROCESS;
END  RT；
```

选择信号赋值语句

```
WITH   表达式   SELECT
       目的信号量 <=  表达式1  WHEN  条件1
                      表达式2  WHEN  条件2
                        ……
                      表达式n  WHEN  条件n;
```

- 选择信号赋值语句类似于CASE语句
- 对表达式进行测试，当表达式取值不同时，将使不同的值代入目标信号量

```
ARCHITECTURE BEHAVE OF MUX4 IS
        SIGNAL SEL: INTEGER;
BEGIN
        WITH SEL SELECT
                Q <=  I0 WHEN 0,
                      I1 WHEN 1,
                      I2 WHEN 2,
                      I3 WHEN 3,
                       'X' WHEN OTHERS;

        SEL <= 0 WHEN A='0'AND B='0' ELSE
               1 WHEN A='0'AND B='1' ELSE
               2 WHEN A='1'AND B='0' ELSE
               3 WHEN A='1'AND B='1' ELSE
               4;
END BEHAVE;
```

并行语句

2.块语句（BLOCK）

- 结构体中的一个子模块可以描述为一个块**BLOCK**
- 块语句把一系列并行语句包装在一起
  - 一个块有它自己的接口
  - 块通过信号和其它的块或端口连接

标号：BLOCK[（保护表达式）]

块头

{说明语句}

BEGIN

{并行语句}；

END BLOCK 标号名；

用于信号、参数的定义和映射，通常通过GENERIC语句、GENERIC MAP语句以及PORT语句和PORT MAP语句来定义接口和传递参数

对该块所用到的对象加以说明，可说明的项目有：USE子句、子程序说明及子程序体、类型说明、常数说明、信号说明、元件说明等

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY ADDER IS
PORT(
        A, B, C_IN : IN STD_LOGIC;
        SUM : OUT STD_LOGIC;
        C_OUT : OUT STD_LOGIC);
END ADDER;
```

```
ARCHITECTURE ADDER_ARCH OF ADDER IS
BEGIN
        EXAMPLE : BLOCK
        PORT(
                A, B, C_IN : IN STD_LOGIC;
                S, C_OUT : OUT STD_LOGIC);
        PORT MAP(A, B, C_IN, SUM, C_OUT);
        BEGIN
                PROCESS(A, B, C_IN)
                BEGIN
                        S <= A XOR B XOR C_IN;
                END PROCESS;
                 C_OUT <= (A AND B) OR (A AND C_IN) OR (B AND C_IN);
        END BLOCK EXAMPLE;
END ADDER_ARCH;
```

并行语句

3.元件说明语句（COMPONENT）

- 在一个设计的结构体中可以使用其它已设计好的元件或模块
- 为了能够调用这些元件或模块，在结构体中必须首先要说明这个元件

元件说明语句的格式

```
COMPONENT 元件名
        GENERIC说明语句;
        PORT说明语句;
END COMPONENT;
```

GENERIC语句用于该元件参数的说明

规定了该元件的输入输出端口

并行语句

4.元件的例化

- 表明在这个结构体中包含了这样一个元件，这个元件有实际的类属值，而且和实际的信号或端口相连
  - 信号或端口的连接关系用端口映射语句**PORT MAP**实现
  - 类属参数的传递用参数映射语句**GENERIC MAP**实现

PORT MAP语句的格式

元件标号：元件名 **PORT MAP**（信号，信号，….）;

- 元件标号在设计中是唯一的
- 下层元件和上层模块之间的连接通过信号之间的对应关系实现
- 信号的映射
  - 位置映射和信号名映射

位置映射

- 在下层元件端口说明中的信号顺序位置和**PORT MAP**语句中指定的实际信号书写顺序位置一一对应

名字映射

把调用模块的端口名称，赋予设计模块中的信号名

二输入与门AND2的端口定义

```
PORT(
        A, B : IN BIT;
        C : OUT BIT);
```

位置映射

u2：AND2 PORT MAP（D1, D2, D3）；

设计中D1对应A，D2对应B，D3对应C

名字映射

u2：AND2 PORT MAP（A => D1, B => D2, C => D3）；

二输入与门AND2

```
ENTITY AND2 IS
GENERIC（RISE, FALL : TIME）;
PORT（
        A, B : IN BIT;
        C : OUT BIT）;
END AND2;
ARCHITECTURE GENERIC_EXAMPLE OF AND2 IS
        SIGNAL INTERNAL_SIGNAL : BIT;
BEGIN
        INTERNAL_SIGNAL <= A AND B;
        C <= INTERNAL_SIGNAL AFTER RISE WHEN
                INTERNAL_SIGNAL = ‘1’ ELSE
                INTERNAL_SIGNAL AFTER FALL;
END GENERIC_EXAMPLE;
```

```
ARCHITECTURE GENERIC_MAP_EXAMPLE OF AND4 IS
        COMPONENT AND2
        GENERIC（RISE, FALL : TIME）;
        PORT（
                A, B : IN BIT;
                C : OUT BIT）;
        END COMPONENT;
        SIGNAL M0, M1 : BIT;
BEGIN
        U0 : AND2
          GENERIC MAP（5 ns, 7 ns）
          PORT MAP(A => D0, B => D1, C => M0);
        U1 : AND2
          GENERIC MAP（5 ns, 7 ns）
          PORT MAP(A => D2, B => D3, C => M1);
        U2 : AND2
          GENERIC MAP（RISE => 9 ns, FALL => 11 ns）
          PORT MAP(A => M0, B => M1, C => Q);
END GENERIC_MAP_EXAMPLE;
```

并行语句

5.生成语句（GENERATE）

- 生成语句GENERATE可以用来产生规则的结构，如块、元件和进程的阵列
  - FOR形式的生成语句
  - IF形式的生成语句

FOR形式的生成语句

```
标号：FOR 变量 IN 离散范围 GENERATE
        ——并行语句
END GENERATE [标号名];
```

FOR形式的生成语句用于描述重复的结构

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY N_ADDER IS
PORT(
            A, B : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            SUM   : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END N_ADDER;
ARCHITECTURE FORGEN_EXAMPLE OF N_ADDER IS
            COMPONENT FULL_ADDER
            PORT(
                        A, B : IN STD_LOGIC;
                        C_IN : IN STD_LOGIC;
                        SUM   : OUT STD_LOGIC;
                        C_OUT : OUT STD_LOGIC);
            END COMPONENT;
            COMPONENT HALF_ADDER1
            PORT(
                        A, B : IN STD_LOGIC;
                        SUM   : OUT STD_LOGIC;
                        C_OUT : OUT STD_LOGIC);
            END COMPONENT;
```

```
        COMPONENT HALF_ADDER2
        PORT(
                A, B : IN STD_LOGIC;
                C_IN : IN STD_LOGIC;
                SUM  : OUT STD_LOGIC);
        END COMPONENT;
        SIGNAL INTER_C : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
        LS_BIT : HALF_ADDER1 PORT MAP(
                        A => A(0),
                        B => B(0),
                        SUM => SUM(0),
                        C_OUT => INTER_C(0));
        GEN_ADDERS : FOR I IN 1 TO 14 GENERATE
                ADDERS : FULL_ADDER PORT MAP(
                        A => A(I),
                        B => B(I),
                        C_IN => INTER_C(I-1),
                        SUM  => SUM(I),
                        C_OUT => INTER_C(I));
        END GENERATE GEN_ADDERS;
        MS_BIT : HALF_ADDER2 PORT MAP(
                        A => A(15),
                        B => B(15),
                        C_IN => INTER_C(14),
                        SUM => SUM(15));
END FORGEN_EXAMPLE;
```

循环变量I也不需要预先定义，在模块中不可见，也不可赋值

IF形式的生成语句

标号：IF 条件 GENERATE

——并行语句

END GENERATE [标号名]；

- IF形式的生成语句用于处理规则结构中的例外情况
  - 比如发生在边界的特殊情况
- 只有在IF条件为真时，才执行结构体内部的语句
- 和顺序语句中的IF语句不同，在IF形式的生成语句中不能含有ELSE语句

```
ARCHITECTURE IFGEN_EXAMPLE OF N_ADDER IS
        COMPONENT FULL_ADDER
        PORT(
                A, B : IN STD_LOGIC;
                C_IN : IN STD_LOGIC;
                SUM  : OUT STD_LOGIC;
                        C_OUT : OUT STD_LOGIC);
        END COMPONENT;
        COMPONENT HALF_ADDER1
        PORT(
                A, B : IN STD_LOGIC;
                SUM  : OUT STD_LOGIC;
                C_OUT : OUT STD_LOGIC);
        END COMPONENT;
        COMPONENT HALF_ADDER2
        PORT(
                A, B : IN STD_LOGIC;
                C_IN : IN STD_LOGIC;
                SUM  : OUT STD_LOGIC);
        END COMPONENT;

        SIGNAL INTER_C : STD_LOGIC_VECTOR(14 DOWNTO 0);
```

```
BEGIN
        GEN_ADDERS : FOR I IN 0 TO 15 GENERATE
                LS_BIT : IF I = 0 GENERATE
                        LS_CELL : HALF_ADDER1 PORT MAP(
                                A => A(0),
                                B => B(0),
                                SUM => SUM(0),
                                C_OUT => INTER_C(0));
                END GENERATE LS_BIT;
                MIDDLE_BITS : IF I>0 AND I<15 GENERATE
                        MIDDLE_CELLS : FULL_ADDER PORT MAP(
                                A => A(I),
                                B => B(I),
                                C_IN => INTER_C(I-1),
                                SUM   => SUM(I),
                                C_OUT => INTER_C(I));
                END GENERATE MIDDLE_BITS;
                MS_BIT : IF I = 15 GENERATE
                        MS_CELL : HALF_ADDER2 PORT MAP(
                                A => A(15),
                                B => B(15),
                                C_IN => INTER_C(14),
                                SUM => SUM(15));
                END GENERATE MS_BIT;
        END GENERATE GEN_ADDERS;
END IFGEN_EXAMPLE;
```