

第10章 包集(Package)和元件(Component)

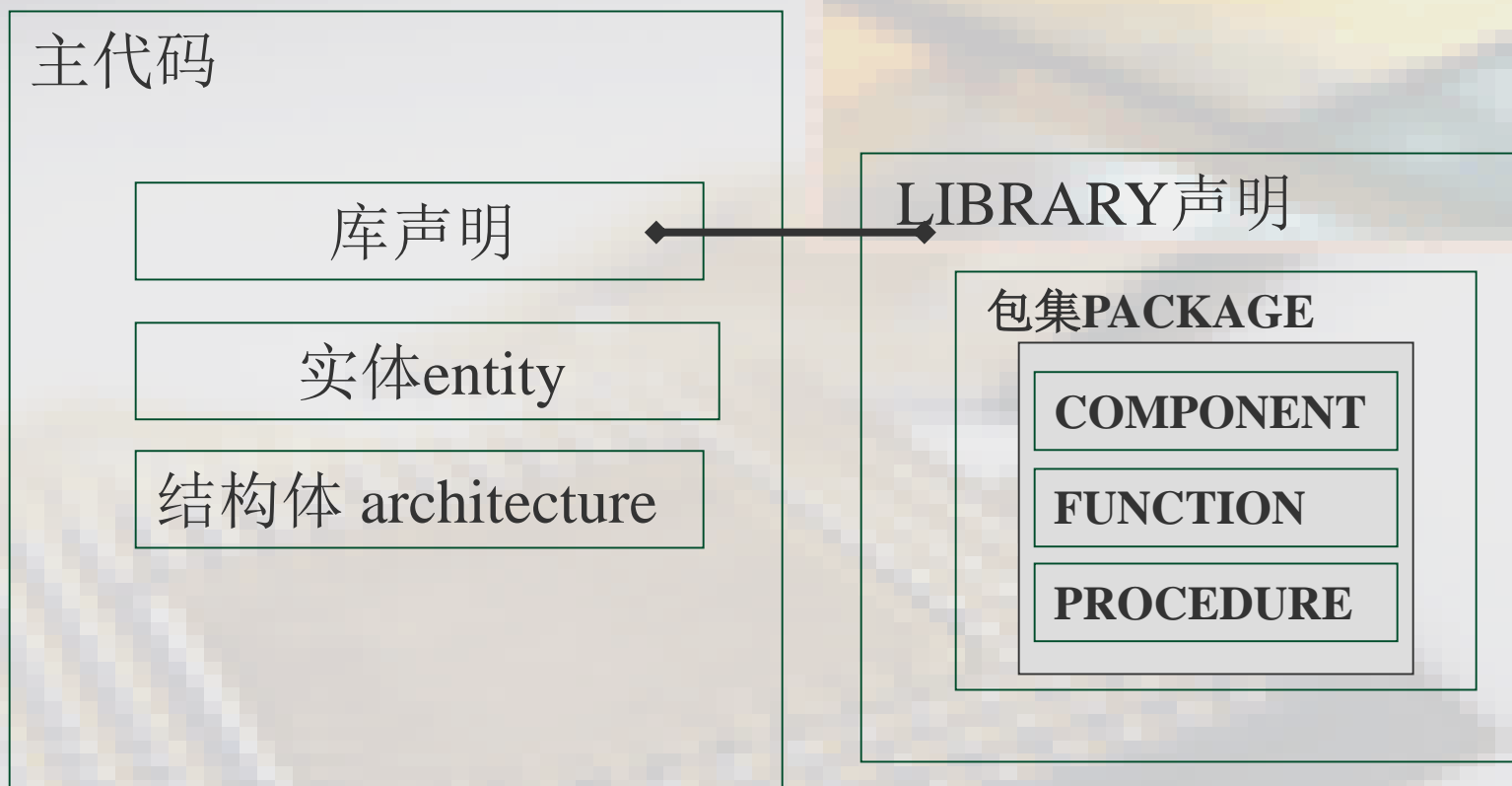
第一部分的内容回顾:

- 代码结构：库的声明、实体、构造体（第2章）；
- 数据类型（第3章）；
- 操作符及属性（第4章）；
- 并发描述与并发代码（第5章）；
- 顺序描述与顺序代码（第6章）；
- 信号、变量与常量（第7章）；
- 有限状态机（第8章）；
- 一些较为简单的电路设计实例（第9章）；

第二部分：通过学习包集、元件、函数与过程，综合运用第一部分知识，力求能够：

- 建立自己的库，使得代码可以被复用或共享；
- 能够设计一些规模较大的电路，并使代码结构清晰。

10.1 概述



VHDL代码的基本组成

10.2 包集 (package)

包集的概念：已定义的常数、数据类型、元件调用说明、子程序的一个集合，类似于C语言中的“.h”文件（头文件）；

使用包集的目的：方便公共信息、资源的访问和共享。

包集的主要内容：除了包含元件(component)、函数(function)和过程(procedure)之外，还可以包含类型(type)和常量(constant)；

元件、函数和过程的作用：经常使用的VHDL代码通常以这几种形式编写，从而可以被添加到包集中，然后被编译到目标library中，有利于代码分割、代码共享和代码重用；

库：多个包集构成库。

包集的语法结构

特殊

包集的声明部分，
必须

```
package 包集名 is
```

```
{ 声明: 元件、函数、过程、类型和常量说明 }
```

```
end 包集名;
```

```
[package body 包集名 is
```

```
{ 包体说明项: 函数、过程的所有描述代码}
```

```
end 包集名; ]
```

包体，可选

注意:

1、虽然包体是可选的，但是如果包集的声明部分有一个或多个函数、过程的声明，则在包体部分必须存在对应的描述代码；如果在包集的声明部分含有元件，则不必在包体中出现该元件的描述代码，元件的代码以完整的.vhd文件存在。

2、package和对应的package body的名称必须相同！

例10.1 简单的程序包

```
library ieee;  
use ieee.std_logic_1164.all;  
package my_package is  
    type state is (st1, st2, st3, st4);  
    type color is (red, green, blue);  
    constant vec: std_logic_vector(7 downto 0):="11111111";  
end my_package;
```

仅包含类型和常量声明的package，不需要使用package body！

例10.2 内部包含函数的package

```
library ieee;  
use ieee.std_logic_1164.all;  
package my_package is  
    type state is (st1, st2, st3, st4);  
    type color is (red, green, blue);  
    constant vec: std_logic_vector(7 downto  
0):="11111111";
```

包集中除了说明类型和变量之外，还包括了一个函数

```
    function positive_edge (signal s: std_logic)  
return boolean;  
end my_package;
```

在包集中出现自定义函数时，须使用package body声明这个函数

```
-----  
package body my_package is  
    function positive_edge(signal s: std_logic)  
return boolean is  
    begin  
        return (s'event AND clk='1');  
    end positive_edge;  
end my_package;
```

package可以被编译成work或其它库中的一部分！

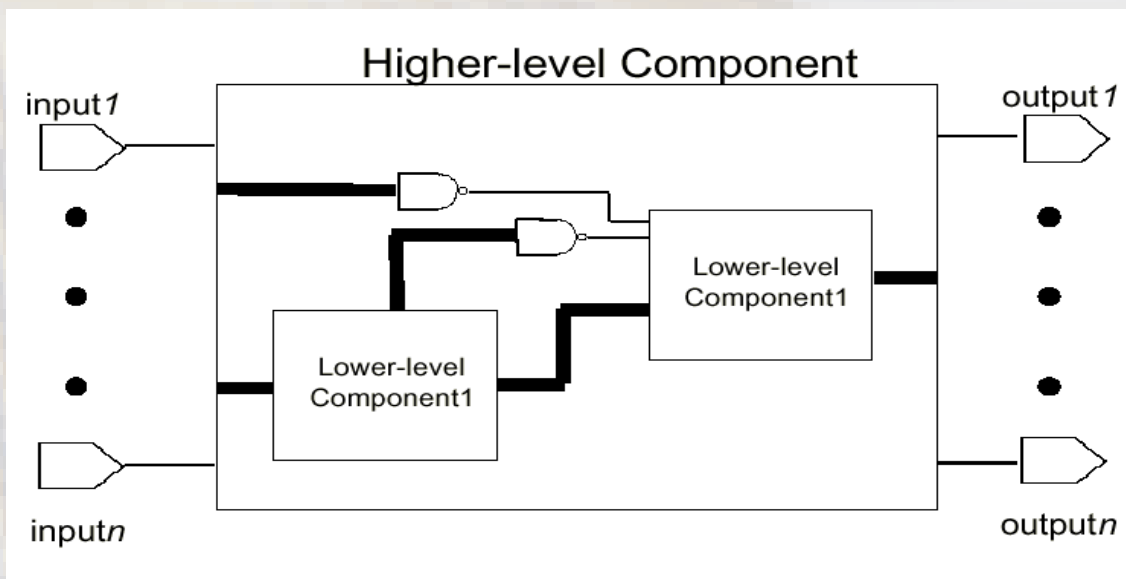
要使用自定义的包集，必须在主程序中加入一个use语句

```
library ieee;  
use ieee.std_logic_1164.all;  
use work.my_package.all;  
-----  
entity .....  
.....  
architecture .....  
.....
```

10.3 元件(component)

一个元件是一段结构完整的代码（包括库声明、实体和结构体这些基本的组成部分），**可以以一个独立的.vhd文件存在。如果将这些代码声明为component，就可以被其它电路或主代码调用，从而使代码具有层次化的结构。**

（函数和过程则只能存放在包集或主代码中，不能以独立的.vhd文件存在并被调用。）



- 元件是一种进行代码分割、代码共享和代码重用的方法。例如，可以将常用的触发器、乘法器、加法器和基本门电路存放为多个.vhd文件，然后将这些代码声明为元件并存放到一个库中，从而可以供所有的设计者方便地调用。
- 在使用（实例化）一个元件之前，必须先对该元件进行声明。
- 元件的声明：

```
component 元件名 IS  
    port (端口声明);  
end component;
```


可在以下部分声明元件：

结构体（Architecture）

包集（Package）

块（Block）

被声明元件的来源：

VHDL设计实体；

其它HDL设计实体；

另外一种标准格式的文件，如EDIF或XNF；

厂商提供的工艺库中的元件、IP核。

■ 元件的实例化(component instantiation)

定义：把低层元件安装（调用）到当前层次设计实体内部的过程。

label: 元件名称

port map (端口列表);

端口列表将元件预定义的端口和实例化时的实际端口关联起来。

例：

以一个反相器为例，该反相器已经完成设计(文件inverter.vhd)并编译到了库work中。
调用过程如下：

-----元件声明-----

```
component inverter is
```

```
    port(a: IN std_logic; b: out std_logic);
```

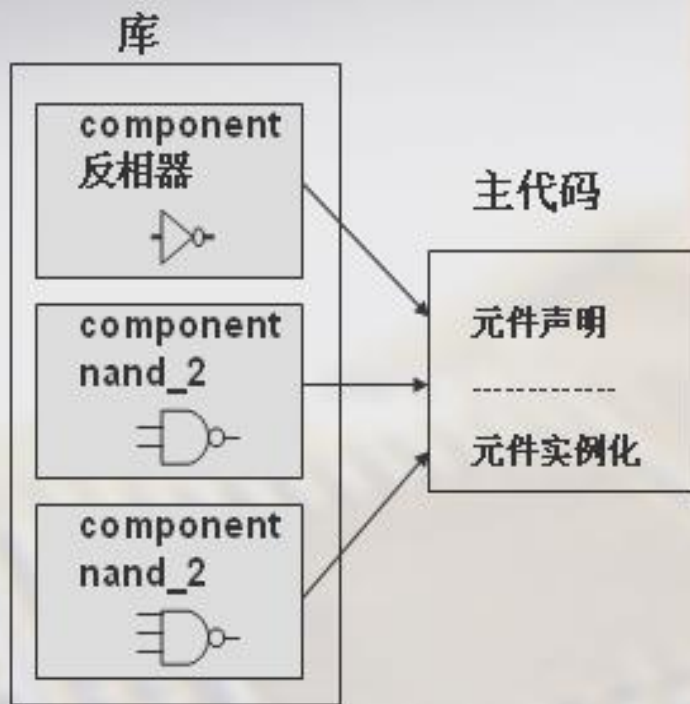
```
end component;
```

-----元件实例化-----

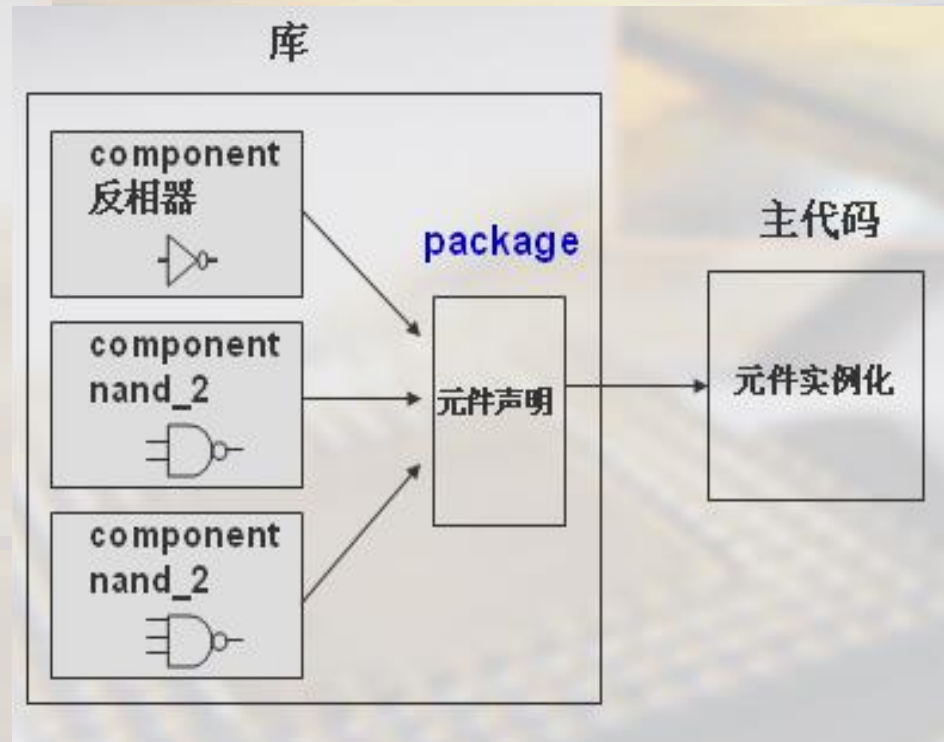
```
u1: inverter port map(x, y);
```

位置映射，端口的排列顺序必须一一对应

元件声明的两种位置（或两种方法）：

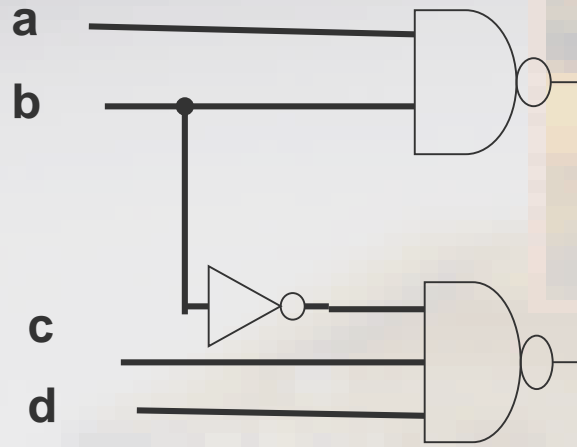


a. 在主代码段中声明



b. 在package中声明

例10.3 在主代码中声明元件



设计步骤:

- 1、编写几个基本组成模块（`inverter` /`nand_2` /`nand_3`）的.vhd文件；
- 2、在主代码中将这几个基本组成模块声明为元件；
- 3、在主代码中实例化这些元件。

具体实现(1): 三个待声明的.vhd文件

-----文件**inverter.vhd**-----

```
library ieee;
use ieee.std_logic_1164.all;
entity inverter is
    port (a: in std_logic;
          b: out std_logic);
end inverter;
architecture inverter of inverter is
begin
    b<= NOT a;
end inverter;
```

-----文件**nand_2.vhd**-----

```
library ieee;
use ieee.std_logic_1164.all;
entity nand_2 is
    port (a, b: in std_logic;
          c: out std_logic);
end nand_2;
architecture nand_2 of nand_2 is
begin
    c<= NOT (a AND b);
end nand_2;
```

-----文件**nand_3.vhd**-----

```
library ieee;
use ieee.std_logic_1164.all;
entity nand_3 is
    port (a, b, c: in std_logic;
          d: out std_logic);
end nand_3;
architecture nand_3 of nand_3 is
begin
    d<= NOT (a AND b AND c);
end nand_3;
```

注意：文件名、实体名必须一致！

具体实现(2): 主.vhd文件, 将前三个文件的代码作为元件加以实例化, 用以实现新的功能。

-----主文件**project.vhd**-----

```
library ieee;
use ieee.std_logic_1164.all;
entity project is
    port (a, b, c, d: in std_logic;
          x, y: out std_logic);
end project;
```

architecture **structural** of **project** is

-----元件在主代码中的声明-----

```
component inverter is
    port (a: in std_logic; b: out std_logic);
end component;    --对应文件inverter.vhd
```

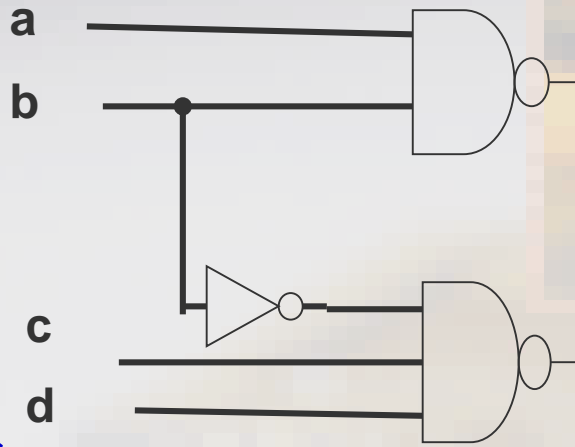
```
component nand_2 is
    port (a, b: in std_logic; c: out std_logic);
end component;    --对应文件nand_2.vhd
```

```
component nand_3 is
    port (a, b, c: in std_logic; d: out std_logic);
end component;    --对应文件nand_3.vhd
```

```
-----
signal w: std_logic;
begin
    u1: inverter port map (b, w);
    u2: nand_2 port map (a, b, x);
    u3: nand_3 port map (w, c, d, y);
end structural;
```

注意: 主文件名与其实体名必须一致, 元件名称必须与元件所在文件的文件名及文件中的实体名一致!

例10.4 在包集中声明元件



设计步骤:

- 1、编写几个基本组成模块（`inverter` /`nand_2` /`nand_3`）的.vhd文件；
- 2、创建一个包集文件，在该包集中将这几个基本组成模块声明为元件；
- 3、在主代码中use该包集；
- 4、在主代码中实例化这些元件。

虽多了一个包集文件，但可避免在主代码中每实例化一个元件就声明一次，在有多项目时很方便！

具体实现：省略前三个.vhd文件

```
-----包集文件my_components.vhd-----  
library ieee;  
use ieee.std_logic_1164.all;  
package my_components is  
    -----元件在包集中的声明-----  
    component inverter is  
        port (a: in std_logic; b: out std_logic);  
    end component;    --对应文件inverter.vhd  
    -----  
    component nand_2 is  
        port (a, b: in std_logic; c: out std_logic);  
    end component;    --对应文件nand_2.vhd  
    -----  
    component nand_3 is  
        port (a, b, c: in std_logic; d: out  
std_logic);  
        end component;    --对应文件nand_3.vhd  
    -----  
  
end my_components;  
-----
```

```
-----主文件project.vhd-----  
library ieee;  
use ieee.std_logic_1164.all;  
use work.my_components.all;  
entity project is  
    port (a, b, c, d: in std_logic;  
        x, y: out std_logic);  
end project;  
architecture structural of project is  
    -----  
    signal w: std_logic;  
begin  
    u1: inverter port map (b, w);  
    u2: nand_2 port map (a, b, x);  
    u3: nand_3 port map (w, x, c, d, y);  
end structural;  
-----
```

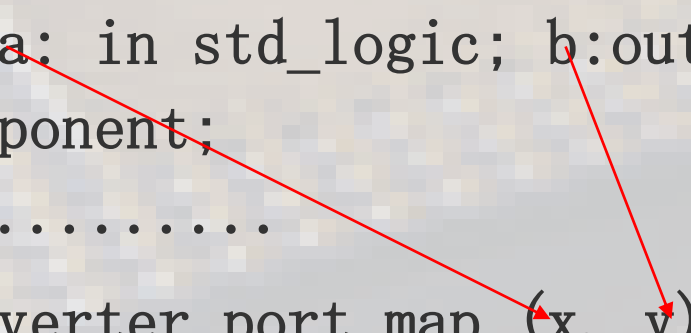
注意：主文件中包集名称必须与包集文件名及包集文件中包集定义名一致！

10.4 端口映射

在元件实例化的过程中，实现元件端口映射有两种方法：**位置映射方法和名称映射方法**。

- **位置映射方法**：例化的端口表达式信号) 必须与元件声明语句中的端口顺序一致。
例：

```
component inverter is
  port (a: in std_logic; b:out std_logic);
end component;
.....
u1: inverter port map (x, y);
```



名称映射方法： 低层次端口名 => 当前层次端口名、信号名。例：

```
component inverter is
  port (a: in std_logic; b: out std_logic);
end component;

.....

u1: inverter port map (x=>a, y=>b);
```

一般说来，位置映射方法书写较为简单，而名称映射方法则不容易出错！

- 对于不需要使用的端口则可以断开，但需使用关键字OPEN，例：

```
u1: inverter port map (x=>a, y=>open, z=>d);
```

10.5 GENERIC参数的映射

在元件进行实例化时如果需要传递参数，则须使用关键字generic（参加4.5节），来进行generic参数的映射。

语法结构：

进行generic参数的
说明和传递

label: 元件名称 generic map(参数列表) port map (端口列表);

第4.5节 通用属性语句（GENERIC）的一种用法

用于指定常规参数，所指定的参数是静态的，方便设计人员进行参数修改，可增加代码的灵活性和可重用性。

- generic语句须在ENTITY中进行声明；
- generic语句所指定的参数是全局的，不仅可以在ENTITY内部使用，还可以在后面的整个设计中使用。
- 主要用来为设计实体指定参数，如端口宽度、器件延时等；

语法结构：

GENERIC （参数名：参数类型:=参数值）；

如：

```
entity my_entity is
```

```
    GENERIC (n:integer:=8);
```

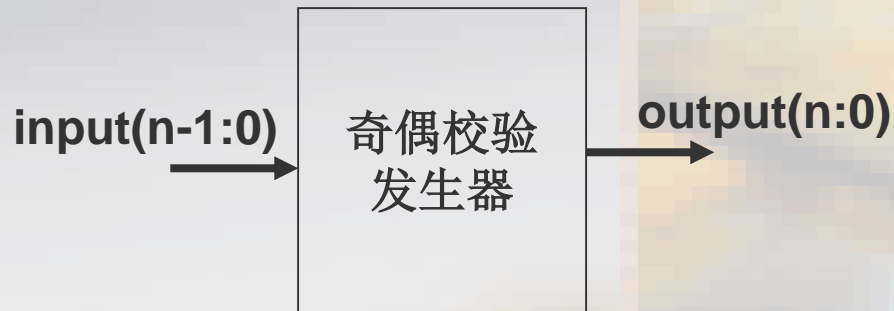
```
    port (...);
```

```
end my_entity;
```

也可以在一个 GENERIC 语句中指定多个参数，如：

```
    GENERIC (n: integer:=8; vector: bit_vector:="1001");
```

例10.5 带有GENERIC参数的元件的实例化



功能:

- 1、输入矢量的宽度比输出矢量的宽度少一位；输出矢量的其它位由输入矢量直接赋值。
- 2、功能固定，规格不固定，可以设计成通用构造体。
- 3、需要统计输入矢量中 '1' 的个数，当输入矢量中 '1' 的个数为奇数时则插入一个 '1'，为偶数时则插入一个 '0'。

设计方案:

- 具有较强的通用性，将输入矢量的宽度 n 作为generic参数加以传递；
- 将代码元件化。

代码实现:

缺省值将被主文件中generic映射时提供的新参数覆盖

-----包集文件 **parity_gen.vhd**-----

```
library ieee;
use ieee.std_logic_1164.all;
entity parity_gen is
    GENERIC (n: integer :=7); --default is 7 ;
    port (input: in bit_vector (n downto 0);
          output: out bit_vector (n+1 downto 0));
end parity_gen;
architecture parity of parity_gen is
begin
    process (input)
        variable temp1: bit;
        variable temp2: bit_vector (output'range);
    begin
        temp1:=0;
        for i IN input'range loop
            temp1:=temp1 XOR input(i);
            temp2(i):=input(i);
        end loop;
        temp2(output'high):=temp1;
        output<=temp2;
    end process;
end parity;
```

须再声明一次，但不必声明默认值

-----主文件 **my_code.vhd**-----

```
library ieee;
use ieee.std_logic_1164.all;
entity my_code is
    GENERIC (n: positive :=2); --2 will overwrite 7 ;
    port (inp: in bit_vector (n downto 0);
          outp: out bit_vector (n+1 downto 0));
end my_code;
architecture my_arch of my_code is
    -----声明元件parity_gen-----
    component parity_gen is
        generic (n: positive) ;
        port (input: in bit_vector (n downto 0);
              output: out bit_vector (n+1 downto 0));
    end component;
    -----
begin
    c1: parity_gen generic map (n) port map (inp, outp);
end my_arch;
```

元件实例化

第10章 思考题

- 1、包集的组成？
- 2、元件的组成、存放的位置、声明的位置？
- 3、端口映射有几种方法？
- 4、generic参数映射有几种？ 实体、元件

作业：

课后习题10.1、10.2；