第11章 函数和过程

函数(function)和过程(procedure)统称为子程序(subprogram)。

- 子程序与进程的相同点：

  内部包含的都是顺序描述代码，使用相同的顺序描述语句，如if,case和loop (wait语句除外)。

- 子程序与进程语句的区别：

  在使用目的上：进程可以直接在主代码中使用；而子程序一般在建库时使用，以便代码重用和代码共享。当然子程序也是可以在主代码中直接建立和使用的。

  在使用方法上：1、子程序不能从结构体的其余部分直接读写信号，所有通信都是通过子程序的接口来完成的；进程可以直接读写结构体内的其它信号。

  2、子程序中不允许使用wait语句。

知识点补充(1)：

子程序的存放位置：

Package、 Architecture、 Process

子程序与硬件规模：

与普通软件中子程序调用的区别：

普通软件子程序调用增加处理时间；

VHDL中每调用一次子程序，其综合后都将对应一个相应的电路模块。子程序调用次数与综合后的电路规模成正比。

设计中应严格控制子程序调用次数。

知识点补充(2)：

子程序的类型：

过程（Procedure）：

0 个或多个 in、inout、

或 out 参数。可获得多个返回值。

函数（Function）：

0 个或多个 in 参数，一个 return 值。

只能获得一个返回值。

过程可作为一种独立的语句结构而单独存在，函数通常作为表达式的一部分来调用。

子程序包含两部分：子程序声明和子程序主体。

知识点补充(3):

每次调用子程序时，都要首先对其进行初始化，即一次执行结束后再调用需再次初始化。因此，子程序内部的值是不能保持的。

11.1 函数(function)

一个函数就是一段顺序描述的代码。对于一些经常遇到的具有共性的设计问题，如数据类型转换、算术运算等，希望将其功能实现的代码被共享和重用，使主代码简洁并易于理解---函数。

注意：由于在于每次调用函数时，都要首先对其进行初始化，即一次执行结束后再调用需再次初始化，即函数内部的值是不能保持的，因此在函数中禁止进行信号声明和元件实例化。

函数的使用过程：先创建函数，再调用函数。

函数的创建：

```
Function   函数名   [<参数列表>] return   数据类型 IS
      [声明]
  BEGIN
     (顺序描述代码)
   END 函数名;
```

参数列表指明函数的输入参数：

- 输入参数的个数是任意的，也可以没有参数；
- 输入参数的类型：变量不能作为参数，可以是常量和信号，语法如下：

```
< (输入)参数列表>= [constant] 常量名：常量类型;
< (输入)参数列表>=SIGNAL 信号名：信号类型;
```

在VHDL语言中，FUNCTION语句只能计算数值，不能改变其参数的值，所以其参数的模式只能是IN，通常可以省略不写。FUNCTION的输入值由调用者拷贝到输入参数中，如果没有特别指定，在FUNCTION语句中按常数或信号处理。因此输入参数不能为变量类型。

另外，由于FUNCTION的输入值由调用者拷贝到输入参数中，因此输入参数不能指定取值范围。

函数的输出：

使用**RETURN**语句，语法结构如下：

```
return  [表达式];
```

return 语句只能用于子程序（函数或过程）中，并用来终止一个子程序的执行。当用于函数时，必须返回一个值。返回值的类型由return后面的数据类型指定。

创建一个函数的例子：

创建名为f1的函数，有3个输入参数a, b, c，其中a, b是常量（关键字constant可以省略），c为信号。输出参数（只能有一个），是boolean类型的。

```
function f1 (a, b: integer; signal c: std_logic_vector) return
boolean IS
  BEGIN
    ( 顺序描述代码 )
  END f1;
```

不能有变量类型

不能指定范围

函数的调用：

函数可以单独构成表达式，也可以作为表达式的一部分被调用。

例：

```
x<=conv_integer(a);      --单独构成表达式；
if x>max(a,b)....        --表达式的一部分；
```

例 11.1： positive_edge（）函数

时钟上升沿检测函数，可用于D触发器的设计：

```
----------函数的创建----------
function positive_edge (signal s: std_logic) return boolean
is
        begin
          return (s'event and s='1');
        end positive_edge;
----------函数的调用----------
.....
if positive_edge (clk) then ....
```

例 11.2： conv_integer（）函数，将std_logic_vector类型的数据转换为integer类型，可处理任意宽度和方向的输入矢量：

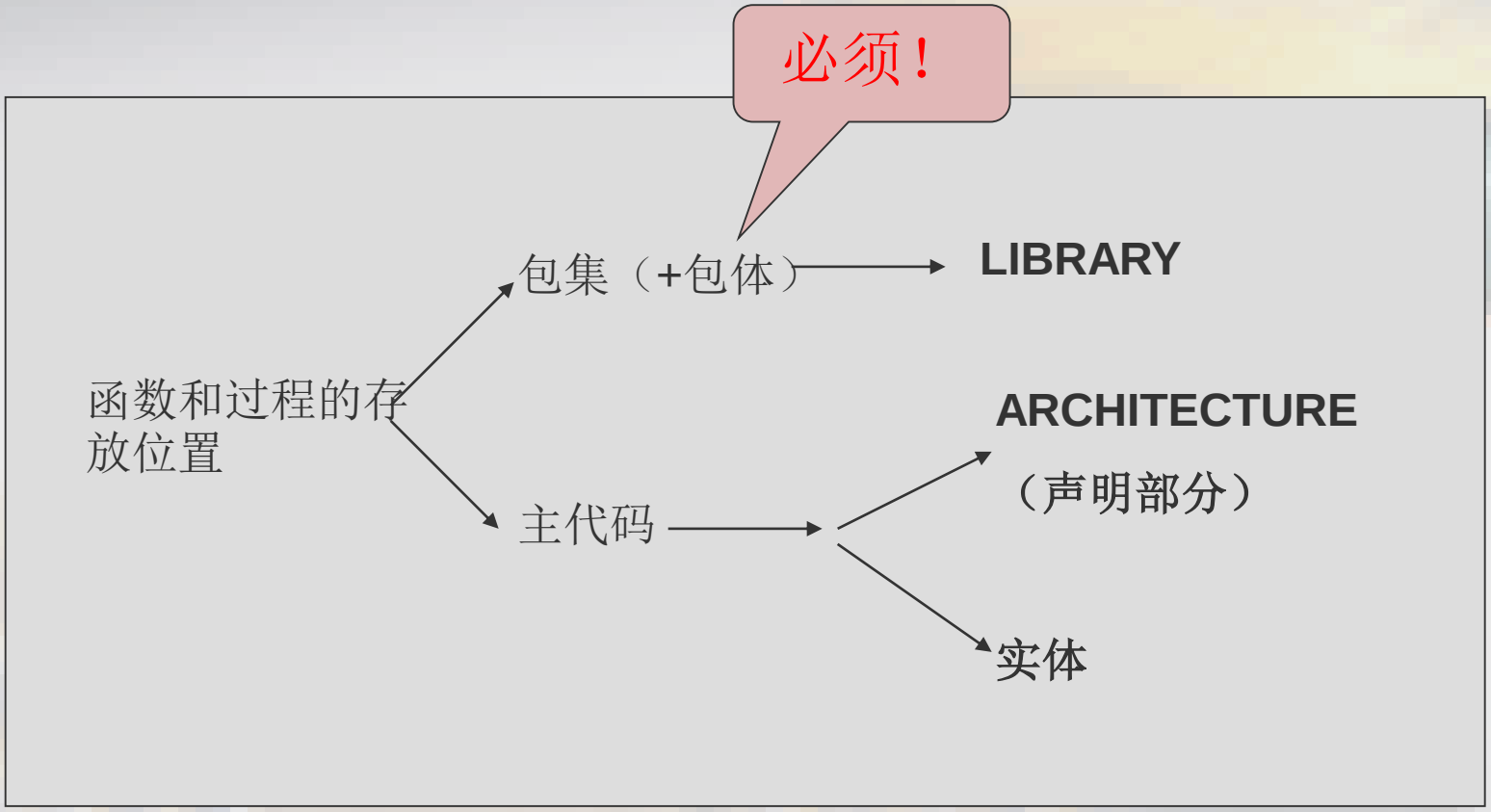
```
----------函数的创建----------
function conv_integer (signal vector: std_logic_vector) return integer is
   variable result: integer range 0 to 2**vector'length-1;
begin
   if (vector(vector'high)='1') then result:=1;
   else result:=0;
   end if;
   for i in (vector'high-1) downto (vector'low) loop
        result:=result*2;
        if (vector(i)='1') then result:=result+1;
        end if;
    end loop;
    return result;
end conv_integer;
----------函数的调用----------
.....
y<=conv_integer (a);

....
```

注意：内部不能声明信号

注意：在输入参数列表中仍然不能指定信号的范围

注意：虽然不知道输入信号的范围，但可以函数被调用时使用s'length来获取输入参数的具体范围

11.2 函数的存放



例 11.3： 在主代码中定义函数，可以出现在entity中，也可以出现在architecture中。如存放于architecture中的声明：

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
    port (d, clk, rst: in std_logic;
          q:out std_logic);
end dff;
architecture my_arch of dff is
    function positive_edge (signal s: std_logic) return boolean is
      begin
       return (s'event and s='1');
      end positive_edge;
begin
    process (clk, rst)
    begin
       if (rst='1') then q<='0';
       else positive_edge(clk) then q<=d;
        end if;
     end process;
end my_arch;
```

例 11.4 (1)： 在包集中定义函数，可以方便地被其它设计所重用和共享。**注意：函数在package中声明，在package body中定义。**

```
----------在包集中定义函数----------
library ieee;
use ieee.std_logic_1164.all;
package my_package is
    function positive_edge (signal s: std_logic) return boolean;
end my_package;
package body my_package is
      function positive_edge(signal s: std_logic) return boolean is
       begin
        return (s'event and s='1');
       end positive_edge;
end my_package;
```

要先声明

再定义

例 11.4(2)： 在主代码中调用在包集中定义的函数。

```
----------在主代码中调用包集中定义的函数----------
library ieee;
use ieee.std_logic_1164.all;
use work.my_package.all;
entity dff  is
     port(d, clk, rst: in std_logic;  q: out std_logic );
end dff;
architecture my_arch of dff is
begin
      process (clk, rst)
      begin
           if (rst='1') then q<='0';
           elsif positive_edge(clk) then q<=d;
           end if;
       end process;
end my_arch;
```

例 11.5： 在包集中定义conv_integer（）函数

```
library ieee;
use ieee.std_logic_1164.all;
package my_package is
  function conv_integer(signal vector: std_logic_vector)
return integer;
end my_package;
package body my_package is
    function conv_integer (signal vector:
std_logic_vector) return integer is
  variable result: integer range 0 to 2**vector'length-1;
begin
  if (vector(vector'high)='1') then result:=1;
  else result:=0;
  end if;
  for i in (vector'high-1) downto (vector'low) loop
      result:=result*2;
      if (vector(i)='1') then result:=result+1;
      end if;
   end loop;
    return result;
 end conv_integer;
end my_package;
```

-------在主代码中调用包集中定义的函数----------

```
library ieee;
use ieee.std_logic_1164.all;
use work.my_package.all;
entity conv_int2  is
    port(a: in std_logic_vector(0 to 3);
         y: out integer range 0 to 15 );
end conv_int2;
architecture my_arch of conv_int2 is
begin
     y<=conv_integer(a);
end my_arch;
```

例 11.6： “+”函数：

- **功能描述：**扩展预定义的“+”操作符，进行std_logic_vector类型数据的加法运算。
- **实现方式：**在包集中定义函数，在主代码中调用。

比较特殊的函数名称

----------在包集中定义函数----------

```
library ieee;
use ieee.std_logic_1164.all;
package my_package is
  function “+” (a, b: std_logic_vector)
return std_logic_vector;
end my_package;
```

```
package body my_package is
    function “+” (a, b: std_logic_vector) return
std_logic_vector is
   variable result: std_logic_vector (3 downto 0);
   variable carry: std_logic;
begin
   carry:=‘0’;
   for i in a’reverse_range loop
        result (i):= a(i) XOR b(i) XOR carry;
         carry:= (a(i) AND b(i)) OR (a(i) AND carry) OR
(b(i) AND carry);
    end loop;
     return result;
 end “+”;
end my_package;
```

-------在主代码中调用包集中定义的函数----------

```
library ieee;
use ieee.std_logic_1164.all;
use work.my_package.all;
entity add_bit  is
     port(a: in std_logic_vector(3 downto 0);
          y: out std_logic_vector( 3 downto 0) );
end add_bit;
architecture my_arch of  add_bit is
    CONSTANT b: std_logic_vector(3 downto 0) :="0011";
    CONSTANT c:  std_logic_vector(3 downto 0) :="0110";
begin
     y<= a + b + c;        --重载 "+"操作符
end my_arch;
```

与通常的函数形式func（<参数列表>）有所不同！

11.3 过程(procedure)

过程与函数的使用目的相似，也是希望将其功能实现的代码被共享和重用，使主代码简洁并易于理解。

过程通常用来定义一个算法，而函数用来产生一个具有特定意义的值；

注意：过程与函数的主要差别就是过程可以有多个返回值。

过程的使用过程：先定义过程，再调用过程。

•过程的定义：

```
PROCEDURE  过程名  [<参数列表>]  IS
      [声明]
  BEGIN
     (顺序描述代码)
   END 函数名;
```

未使用return语句

参数不仅仅是IN类型了！

参数列表指明过程的输入和输出参数：

•参数的个数是任意的；

•参数的模式：输入(IN)、输出(OUT)、双向(INOUT)；

•参数的类型：变量、常量和信号；

•输入模式下，默认的参数类型是常量；

输出模式和双向模式下，默认的参数类型是变量；

< **(IN/OUT/INOUT)**参数列表>= **[constant]** 常量名：**模式**类型；

< **(IN/OUT/INOUT)**参数列表>=**SIGNAL** 信号名：**模式**类型；

< **(IN/OUT/INOUT)**参数列表>=**VARIABLE** 变量名：**模式**类型；

- 与PROCESS相同的是，过程结构中的语句也是顺序执行的。调用者在调用过程前应先将初始值传递给过程的输入参数，然后启动过程语句，按顺序自上至下执行过程结构中的语句，执行结束，将输出值拷贝到调用者制定的变量或信号中。
- 过程内部的WAIT语句、信号声明和元件调用都是不可综合的。
- 一个可综合的过程内部不能包含或隐含寄存器。

例：

省略了关键字CONSTANT

```
procedure my_procedure (a: in bit;
                        signal b, c: in bit;
                        signal x: out bit_vector (7 downto 0);
                        signal y: inout integer range 0 to 99)   is
begin
        .....
end my_procedure;
```

过程调用：

过程可作为独立的语句被调用。

例：

----------直接进行过程调用----------

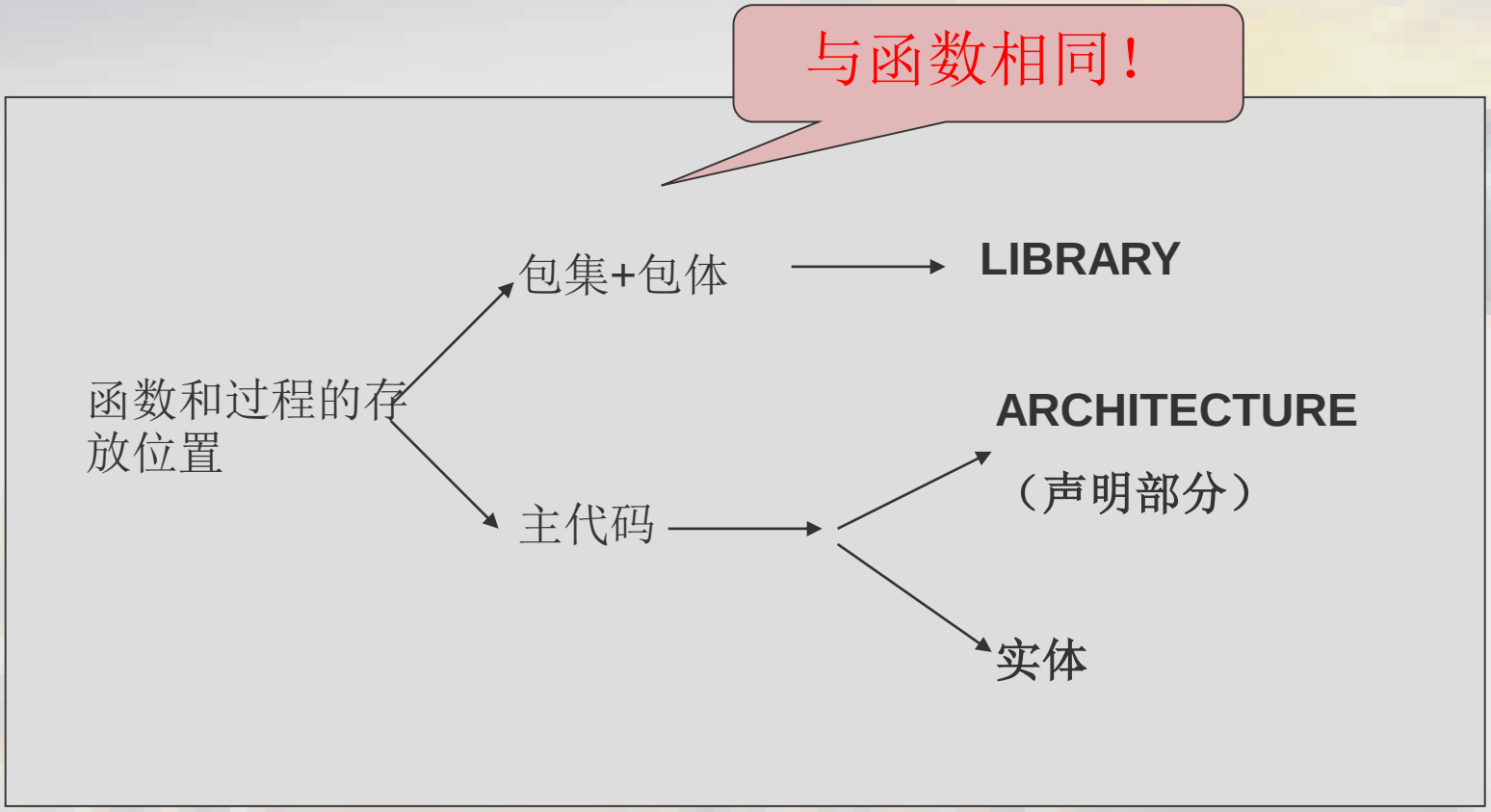
compute_min_max(in1, in2, in3, out1, out2);

divide(dividend, divisor, quotient, remainder);

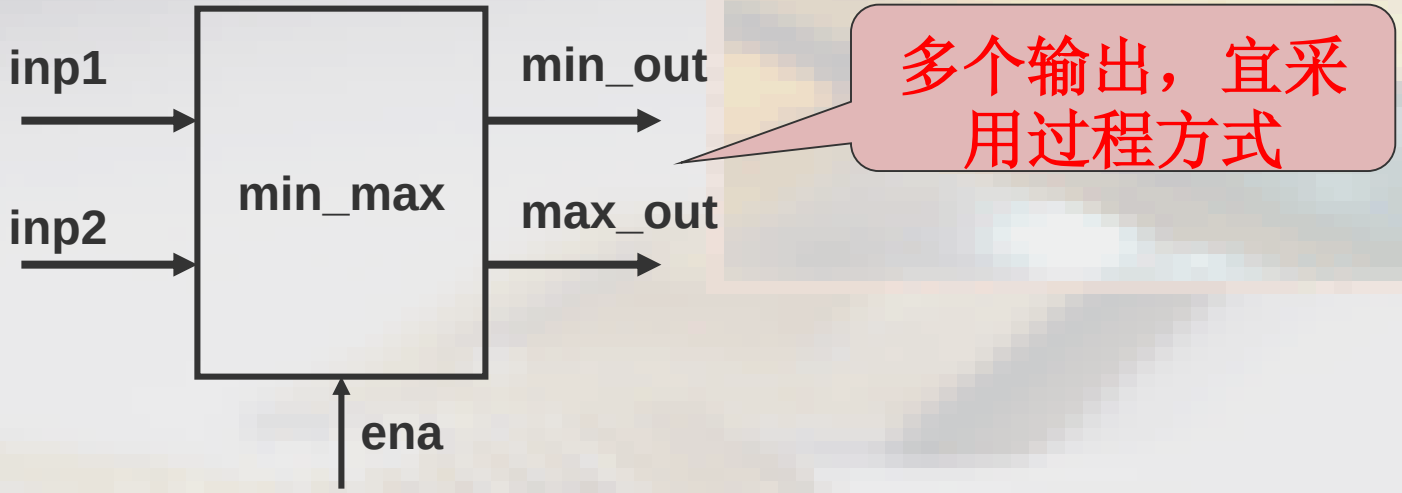
----------在其他语句中调用过程----

if (a>b) then proc_add(in1, in2, out_carry, out_sum) ;

11.4 过程的存放



例11.9：过程存放在主代码中



过程sort的功能描述：对输入两个8位的无符号整数进行比较，数值小的从min_out输出，数值大的从max_out输出。

代码实现：　--在主代码中存放过程

```
library ieee;
use ieee.std_logic_1164.all;
entity min_max is
    generic(limit: integer :=255);
     port ( ena: in bit;       inp1, inp2: in integer range 0 to limit;
            min_out, max_out: out integer range 0 to limit);
end min_max;
architecture my_arch of min_max  is
     procedure sort (signal in1, in2: in integer range 0 to limit;
                     signal min, max: out  integer range 0 to limit)  is
        begin
            if (in1>in2) then   max<=in1; min<=in2;
            else          max<=in2; min<=in1;
            end if;
        end sort;
begin
      process (ena)
      begin
          if (ena='1') then sort (inp1, inp2, min_out, max_out);
          end if;
       end process;
end my_arch;
```

名称可以不同，但顺序要一致！

例11.10：过程存放在包集中 （续前例）

```
-------在包集中定义过程sort------------
library ieee;
use ieee.std_logic_1164.all;
PACKAGE my_package is
    CONSTANT  limit: integer :=255);
    PROCEDURE sort (signal in1, in2: in integer range 0 to limit;
                            signal min, max: out integer range 0 to limit);
END PACKAGE;
PACKAGE BODY my_package IS
      procedure sort (signal in1, in2: in integer range 0 to limit;
                          signal min, max: out  integer range 0 to limit)  is
       begin
           if (in1>in2) then   max<=in1; min<=in2;
           else          max<=in2; min<=in1;
           end if;
       end sort;
END PACKAGE;
```

```
----------------在主代码中调用包集中的过程sort-----------------
library ieee;
use ieee.std_logic_1164.all;
USE work.my_package.all;
entity min_max1 is
    generic(limit: integer :=255);
     port ( ena: in bit;
               inp1, inp2: in integer range 0 to limit;
              min_out, max_out: out integer range 0 to limit);
end min_max1;
architecture my_arch of min_max1  is
begin
      process (ena)
      begin
          if (ena='1') then sort (inp1, inp2, min_out, max_out);
          end if;
       end process;
end my_arch;
```

11.5 函数与过程小结

- 函数有零个或多个输入（模式）参数和一个返回值（return 语句），输入参数只能是常量（默认）或信号，不能被改变，不能是变量。
- 过程可以具有多个输入/输出/双向模式的参数，可以是信号、变量和常量；对输入模式的参数，默认的为常量（不可改变），对于输出和双向模式的参数，默认的为变量（可变）（无需使用return语句）；
- 函数调用是作为表达式一部分出现的，过程则可以直接调用；
- 函数和过程内部的wait和component都是不可综合的；
- 两者的存放位置相同。

11.6 断言(ASSERT)语句

- 断言语句的作用：将仿真过程中出现的问题通过屏幕等方式提示出来，可用于调试代码。
- 因为无法对应具体的电路，因此断言语句是不可综合的。
- 断言语句所提示问题的严重程度：

  attention/warning/error/failure,根据问题的严重程度，仿真过程可被终止。
- 断言语句的语法格式：

```
ASSERT condition
  [REPORT “message”]
  [SERVERITY severity_level];
```

当condition为false时，显示message！

例： 检测一个加法函数中两个输入参数是否具有相同的位宽时，在函数体内插入assert语句：

```
ASSERT a'length=b'length

report "error:vectors do not have same
length!"

severity failure;
```

注意：assert语句不会消耗硬件资源，也不可综合！

第11章　思考题

1、使用函数与过程的目的是什么？

2、函数的输入参数有何特点，怎样输出结果？

3、过程的输入/输出有何特点？

4、函数与过程的存放与调用特点？

作业：

1、课后习题11.1；

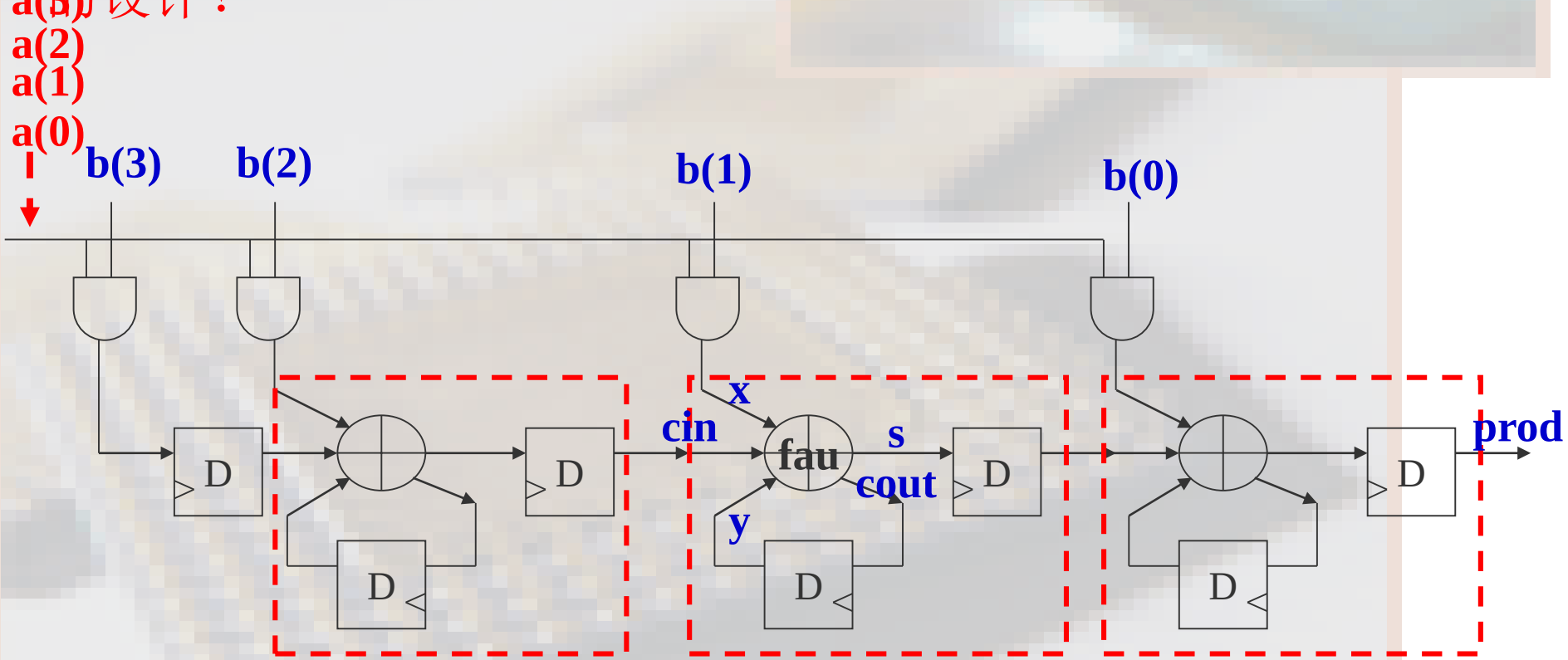
2、读懂例11.7、11.8；

3、预习第12章，准备上机；

例1、串-并型乘法器

功能描述：输入矢量a以串行的方式，从低位开始，逐位地输送到运算电路中。另一个输入矢量b（4位）的所有位以并行方式同时输入。

流水线结构！ 流水线单元的设计！



代码实现：

结构化的描述方法：整个设计由多个元件组合而成。

-------文件**and_2.vhd(component)** ----------

```
library ieee;
use ieee.std_logic_1164.all;
entity and_2 is
    port (a,b: in std_logic;
          y: out std_logic);
end and_2;
architecture and_2 of and_2 is
begin
     b<=  a AND b;
end and_2;
-----------------------------------------
```

----------文件**reg.vhd (component)** ---------

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
    port (d, clk, rst: in std_logic;
          q: out std_logic);
end reg;
architecture reg of reg is
begin
     process (clk, rst)
     begin
          if (rst='1') then q<='0';
          elsif (clk'event and clk ='1') then
              q<=d;
          end if;
     end process;
end reg;
-----------------------------------------
```

-------文件**fau.vhd(component)** ---------

```
library ieee;
use ieee.std_logic_1164.all;
entity fau is
   port (a, b, cin: in std_logic;
         s, cout: out std_logic);
end fau;
architecture fau of fau is
begin
     s<=  a XOR b XOR cin;
     cout<=(a AND b) OR (a AND cin) OR
           (b AND cin);
end fau;
-----------------------------------------
```

-------文件**pipe.vhd(component)** ---------

```
library ieee;
use ieee.std_logic_1164.all;
Use work.my_components.all;
entity pipe is
   port (a, b, clk, rst: in std_logic;
         q: out std_logic);
end pipe;
architecture structural of pipe is
    signal s, cin, cout: std_logic;
begin
     u1: component fau port map (a, b, cin, s, cout);
     u2: component reg port map (cout, clk, rst, cin);
     u3: component reg port map (s, clk, rst, q);
end structural;
-----------------------------------------
```

---文件**my_components.vhd(package)** -------

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

PACKAGE my_components IS
    component and_2 IS
        port (a, b: IN std_logic;  y: out std_logic);
    end component;

    component fau IS
        port (a, b, cin: in std_logic;
                s, cout: out std_logic);
    end component;

    component reg IS
         port (d ,clk, rst: in std_logic;
                 q: out std_logic);
    end component;

    component pipe is
        port (a, b, clk, rst: in std_logic;
                q: out std_logic);
    end component;

End my_components;
```

---文件**multiplier.vhd(project)** -------

```
library ieee;
use ieee.std_logic_1164.all;
Use work.my_components.all;

Entity multiplier is
        port (a, clk, rst: IN std_logic;
                b: in std_logic_vector(3 downto 0);
                prod: out std_logic);
end multiplier;

Architecture structural of multiplier is
   signal and_out, reg_out: std_logic_vector(3 downto 0);
Begin
   u1: component and_2 port map (a, b(3), and_out(3));
   u2: component and_2 port map (a, b(2), and_out(2));
   u3: component and_2 port map (a, b(1), and_out(1));
   u4: component and_2 port map (a, b(0), and_out(0));
   u5: component reg port map (and_out(3), clk, rst, reg_out(3));
   u6: component pipe port map (and_out(2), reg_out(3),   clk, rst, reg_out(2));
   u7: component pipe port map (and_out(1), reg_out(2),   clk, rst, reg_out(1));
   u8: component pipe port map (and_out(0), reg_out(1),   clk, rst, reg_out(0));
  prod<=reg_out(0);
End structural;
-----------------------------------------
```

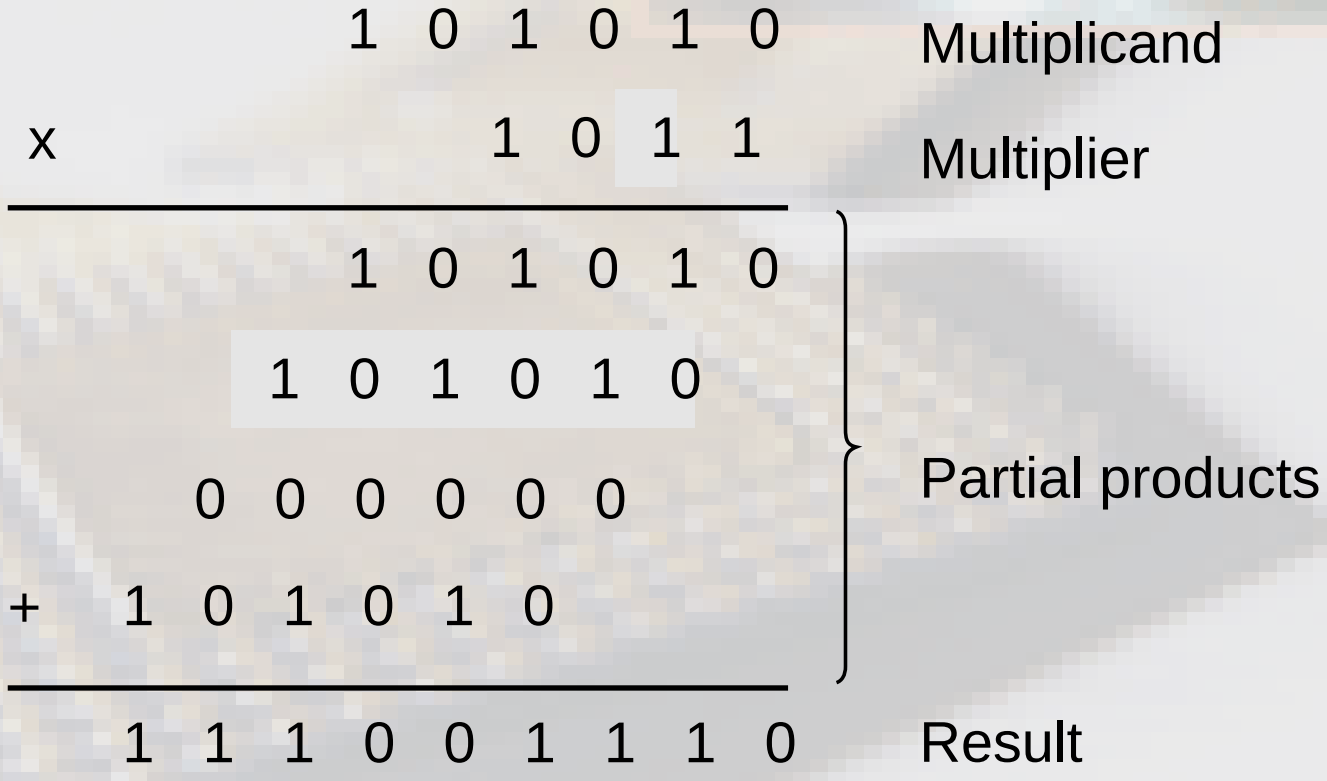
几个问题：

- 输入/输出的延时关系？如从a（0）输入，到第一个prod值输出需要多长的时间？（1个clk周期，等于b的位宽）
- 一个完整的工作周期是多少？（a的位宽+b的位宽个clk周期，这里是8个clk周期）
- 测试矢量的选择：0000、1111、高阻？
- 测试时序的选择：1个clk、2个clk、3个clk、8个clk、8*n个clk？

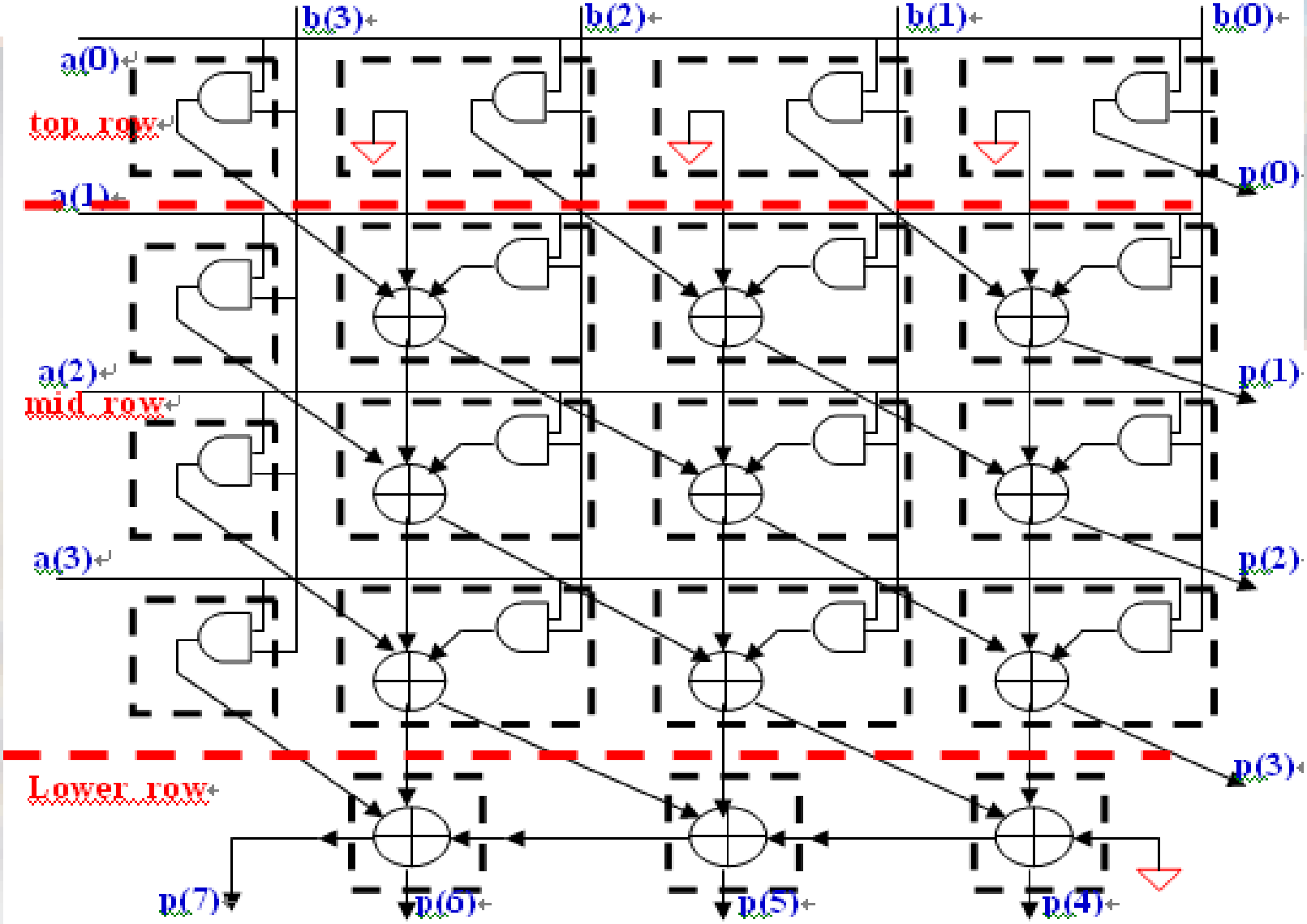
例2、并行乘法器

功能描述：输入矢量a、b以并行的方式输送到乘法运算电路中。无需使用寄存器存储中间数据。 结果为prod。

非流水线结构！无需使用时钟信号！



将P（i）扳直了看！


b(3)
b(2)
b(1)
b(0)
a(0)
top row
a(1)
p(0)
a(2)
mid row
p(1)
a(3)
p(2)
p(3)
Lower row
p(7)
p(6)
p(5)
p(4)

设计思路：

- 基于元件实例化；直接使用例1的包集元件设计文件and_2和fau；
- 分为三排：top_row, mid_row, lower_row；
- 自定义包集，包含将要用到的元件；
- 主代码multiplier调用这些元件；
- 全部采用fau，至于一些至于两个输入的加法操作，则通过输入一个‘0’（接地）来作为fau的一个输入端。（取代hau）

-------文件**lower_row.vhd(component)** ------

```
library ieee;
use ieee.std_logic_1164.all;
Use work.my_components.all;
entity lower_row is
    port (sin, cin: in std_logic_vector(2 downto 0);
          p: out std_logic_vector(3 downto 0));
end lower_row;
architecture structural of lower_row is
     signal local: std_logic_vector(2 downto 0);

begin
      local(0)<='0';
 u1: component fau port map(sin(0), cin(0),
local(0), p(0), local(1) );
 u2: component fau port map(sin(1), cin(1),
local(1), p(1), local(2) );
u3: component fau port map(sin(2), cin(2),
local(2), p(2), p(3));
end structural;
```

三级串行fau相连

-------文件**top_row.vhd(component)** ------

```
library ieee;
use ieee.std_logic_1164.all;
Use work.my_components.all;
entity top_row is
    port (a: in std_logic;
          b: in std_logic_vector(3 downto 0);
          sout, cout: out std_logic_vector(2 downto 0);
          p: out std_logic);
end top_row;
architecture structural of top_row is
begin
u1: component and_2 port map(a, b(3), sout(2));
u2: component and_2 port map(a, b(2), sout(1));
u3: component and_2 port map(a, b(1), sout(0));
u4: component and_2 port map(a, b(0), p);
Cout(2)<='0';
Cout(1)<='0';
Cout(0)<='0';
end structural;
```

-------文件**mid_row.vhd(component)** ------

```
library ieee;
use ieee.std_logic_1164.all;
Use work.my_components.all;
entity mid_row is
   port (a: in std_logic;
         b: in std_logic_vector(3 downto 0);
         sin, cin: in std_logic_vector(2 downto 0);
     sout, cout: out std_logic_vector(2 downto 0);
         p: out std_logic);
end mid_row;
architecture structural of mid_row is
   signal and_out: std_logic_vector(2 downto 0);
begin
u1: component and_2 port map(a, b(3), sout(2));
u2: component and_2 port map(a, b(2), and_out(2));
u3: component and_2 port map(a, b(1), and_out(1));
U4:component and_2 port map(a, b(0), and_out(0));
u5: component fau port map(sin(2), cin(2),
and_out(2), sout(1), cout(2));
u6: component fau port map(sin(1), cin(1),
and_out(1), sout(0), cout(1));
u7: component fau port map(sin(0), cin(0),
and_out(0), p, cout(0));
end structural;
-----------------------------------------
```

---文件**my_components.vhd(package)** -------

```
library ieee;
use ieee.std_logic_1164.all;
PACKAGE my_components IS

    component and_2 IS
        port (a, b: IN std_logic;  y: out std_logic);
    end component;

    component fau IS
        port (a, b, cin: in std_logic;
              s, cout: out std_logic);
    end component;

    component top_row IS
         port (a: in std_logic;
               b: in std_logic_vector(3 downto 0);
    sout, cout: out std_logic_vector(2 downto 0);
               p: out std_logic);
    end component;

    component mid_row is
          port (a: in std_logic;
                b: in std_logic_vector(3 downto 0);
     sin, cin: in std_logic_vector(2 downto 0);
     sout, cout: out std_logic_vector(2 downto 0);
                p: out std_logic);
      end component;

      component lower_row IS
          port (sin, cin: in std_logic_vector(2
downto 0);
                p: out std_logic_vector(3 downto 0));
      end component;
End my_components;
-----------------------------------------
```

```
---主文件multiplier.vhd(project) -------
library ieee;
use ieee.std_logic_1164.all;
Use work.my_components.all;

Entity multiplier is
        port (a, b: IN std_logic_vector(3 downto 0);
                prod: out std_logic_vector(7 downto 0));
end multiplier;

Architecture structural of multiplier is
    type matrix is array (0 to 3) of std_logic_vector (2 downto 0);
    signal s, c: matrix;
Begin
    u1: component top_row port map (a(0), b, s(0), c(0), prod(0));
    u2: component mid_row port map (a(1), b, s(0), c(0), s(1), c(1), prod(1));
    u3: component mid_row port map (a(2), b, s(1), c(1), s(2), c(2), prod(2));
    u4: component mid_row port map (a(3), b, s(2), c(2), s(3), c(3), prod(3));
   u5: component lower_row port map (s(3), c(3),   prod(7 downto 4));
End structural;
-----------------------------------------------
```

课后习题

- 复习例12.1和例12.2