

VHDL培训教程

欢迎参加VHDL培训

浙江大学电子信息技术研究所
电子设计自动化(EDA)培训中心

编写:王勇 TEL:7951949或7951712

EMAIL:wangy@isee.zju.edu.cn

VHDL培训教程

第一讲、VHDL简介及其结构

第二讲、VHDL中的对象、操作符、数据类型

第三讲、VHDL中的控制语句及模块

第四讲、状态机的设计

第一讲、VHDL简介及其结构

- 通过本课的学习您可以了解以下几点
 - 1、VHDL的基本概念
 - 2、VHDL的基本结构
 - 3、VHDL的设计初步

什么是VHDL

- VHDL-

VHSIC Hardware Description Language

其中VHSIC-

Very High Speed Integrated Circuit

电子设计自动化的关键技术之一一是要求用形式化方法来描述硬件系统。VHDL适应了这种要求。

VHDL和Verilog HDL

- Verilog HDL:

另一种硬件描述语言，由Verilog 公司开发，1995年成为IEEE标准。

优点：简单、易学易用

缺点：功能不如VHDL强大，仿真工具少

- VHDL :

1987年成为IEEE标准

优点：功能强大、通用性强。

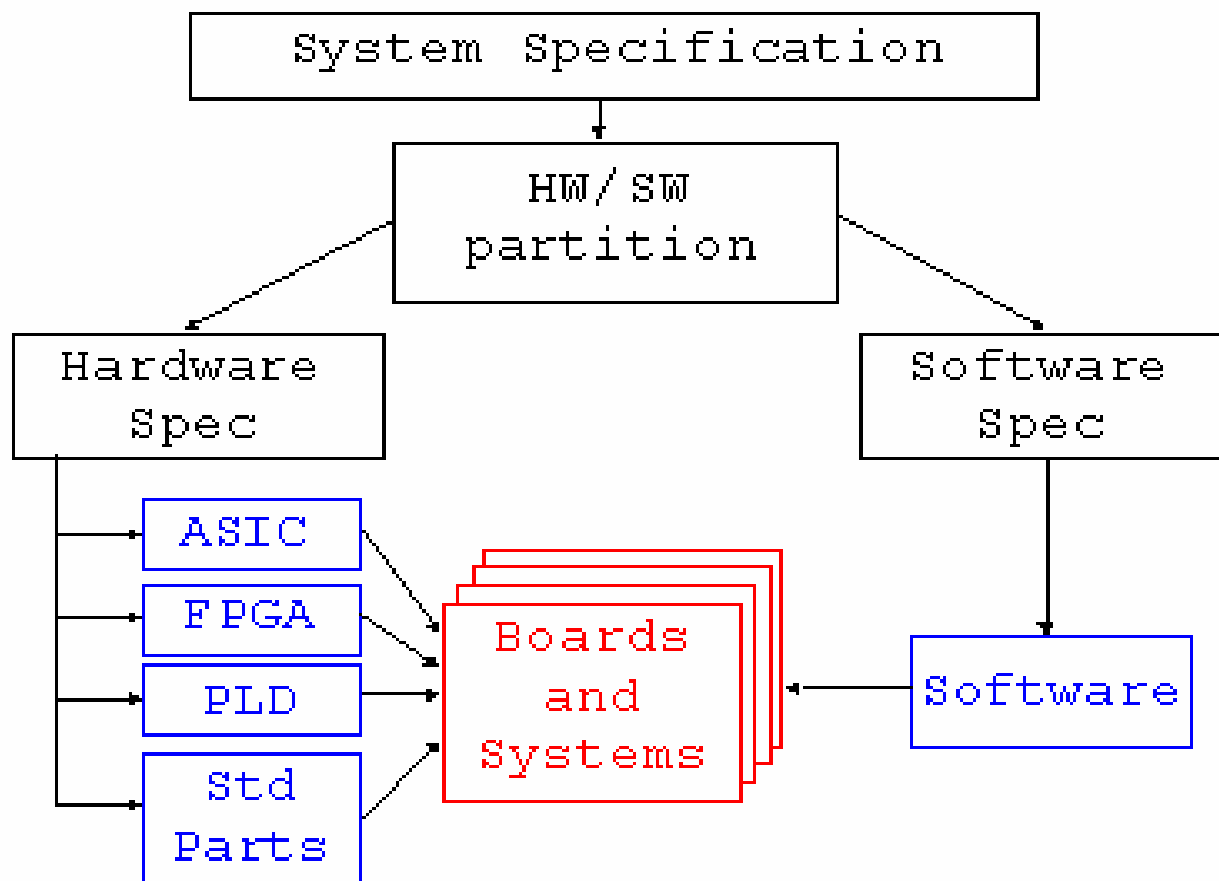
缺点：难学

VHDL的发展历史

- 起源于八十年代，由美国国防部开发
- 两个标准：
 - 1、1987年的 IEEE 1076 (VHDL87)
 - 2、1993年进行了修正 (VHDL93)

VHDL在电子系统设计中的应用

- 电子系统的设计模块

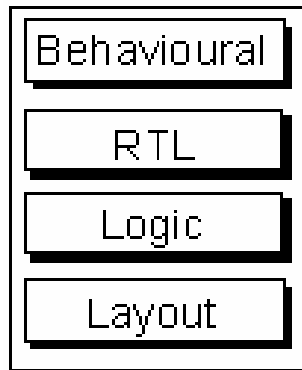


VHDL在电子系统设计中的应用

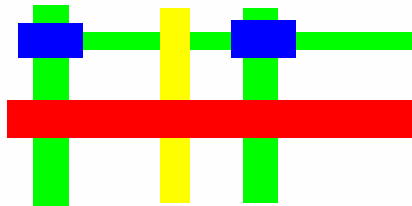
- 电子系统设计的描述等级
 - 1、行为级
 - 2、RTL级（Register transfer level）
 - 3、逻辑门级
 - 4、版图级
- 用VHDL可以描述以上四个等级

VHDL在电子系统设计中的应用

- 系统设计的描述等级-制版级



 **Layout Level**

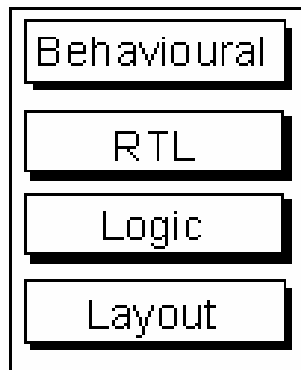


Layout on silicon

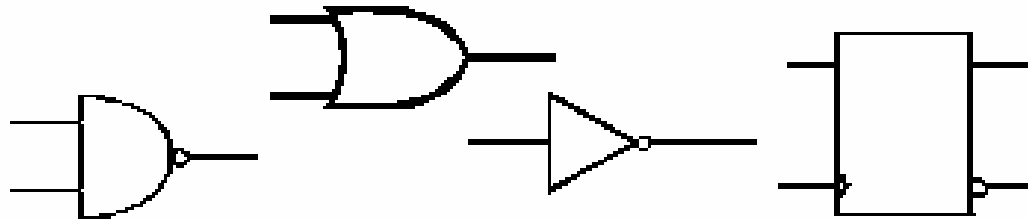
Timing, analog effects

VHDL在电子系统设计中的应用

- 系统设计的描述等级-逻辑门级



 **Logic Level**

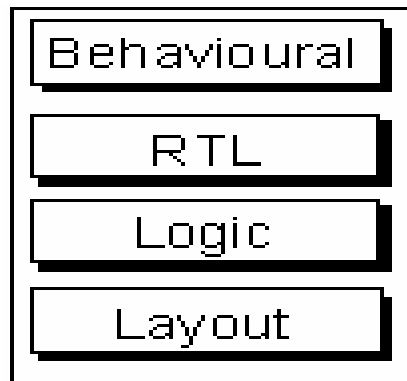


Function, architecture, technology

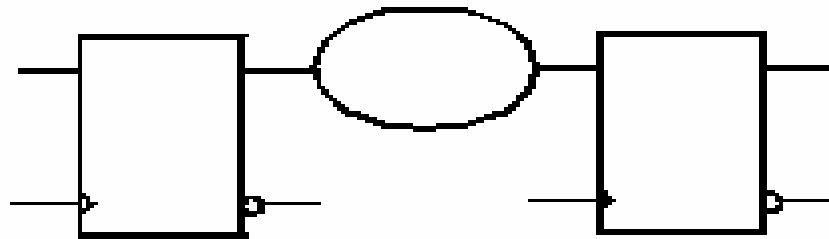
Detailed timing

VHDL在电子系统设计中的应用

- 系统设计的描述等级-RTL级



**Register
Transfer Level**

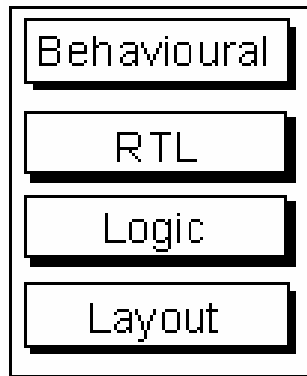


Cycle based timing

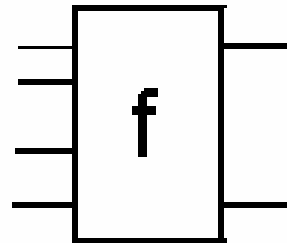
Function and register architecture

VHDL在电子系统设计中的应用

- 系统设计的描述等级-行为级



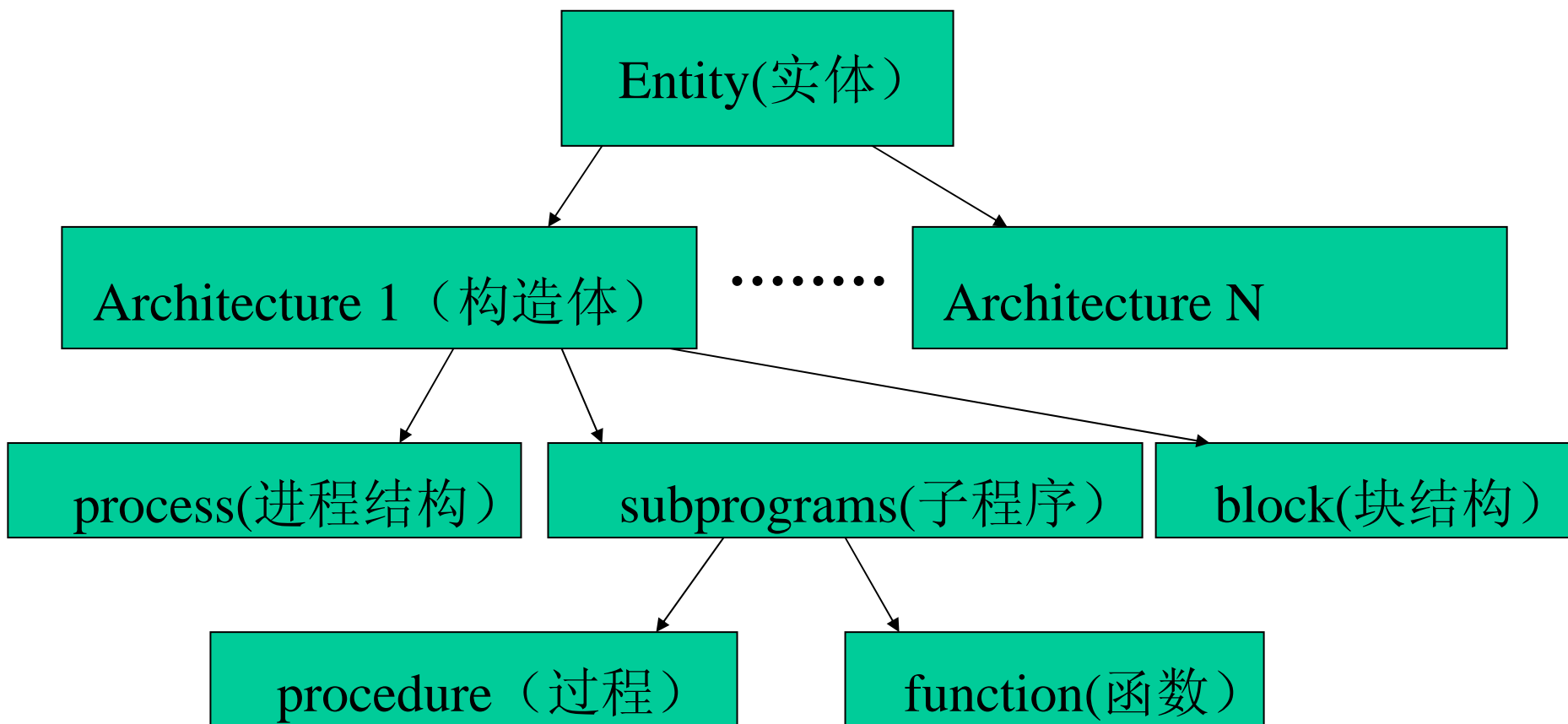
Behavioural



Function only, no architecture

Timing detail as required

如何使用VHDL描述硬件实体



```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.std_logic_unsigned.all;
```

```
entity count is
```

```
    port ( clock,reset: in STD_LOGIC;  
          dataout: out STD_LOGIC_VECTOR (3 downto 0) );
```

```
end count;
```

```
architecture behavior1 of count is
```

```
    signal databuffer: STD_LOGIC_VECTOR (3 downto 0);
```

```
begin
```

```
    dataout<=databuffer;
```

```
    process(clock,reset)
```

```
    begin
```

```
        if (reset='1') then    databuffer<="0000";
```

```
        elsif (clock'event and clock='1') then
```

```
            if databuffer="1000" then
```

```
                databuffer<="0000"; else    databuffer<=databuffer+'1';
```

```
            end if;
```

```
        end if;
```

```
    end process;
```

```
end behavior1;
```

VHDL结构要点

1、ENTITY（实体）

格式：

```
Entity 实体名 IS  
    [类属参数说明]  
    [端口说明]  
End Entity;
```

其中端口说明格式为：

PORT(端口名1, 端口名N: 方向: 类型)

其中方向有: IN , OUT, INOUT, BUFFER, LINKAGE

VHDL结构要点

- 注意

In 信号只能被引用,不能被赋值
out 信号只能被赋值,不能被引用
buffer 信号可以被引用,也可以被赋值

- 简单地说

In 不可以出现在 \leq 或 $:=$ 的左边
out 不可以出现在 \leq 或 $:=$ 的右边
buffer 可以出现在 \leq 或 $:=$ 的两边

VHDL结构要点

- 例子 (HalfAdd)

```
entity HALFADD is  
  port (A,B : in bit;  
        SUM, CARRY : out bit);  
end HALFADD;
```



其内部结构将由Architecture来描述

VHDL结构要点

2、Arcthitecture（构造体）

格式：

Arcthitecture 构造体名 **of** 实体名 **is**

[定义语句] 内部信号、常数、元件、数据类型、函数等的定义

begin

[并行处理语句和block、process、function、procedure]

end 构造体名；

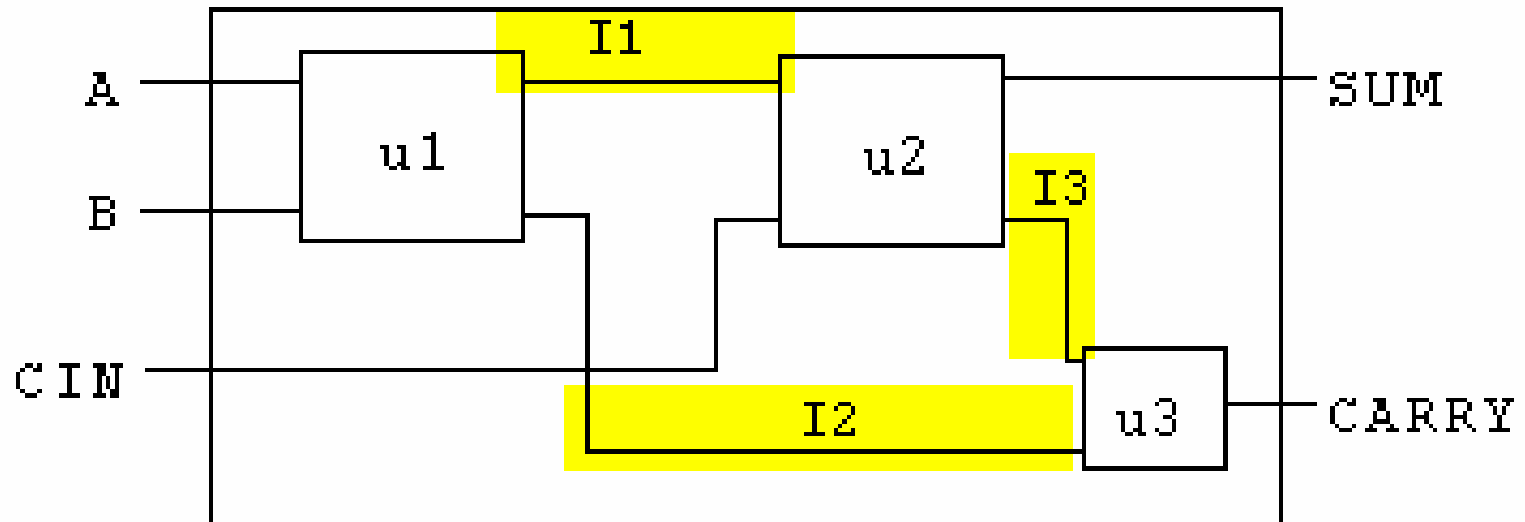
VHDL结构要点

- 例子(HalfAdd)

```
architecture BEHAVE of HALFADD is
begin
    SUM <= A xor B;
    CARRY <= A and B;
end BEHAVE;
```

VHDL结构要点

- 例子 (FullAdd) (学习如何调用现有模块)



VHDL结构要点

- 实例(FullAdd)-entity

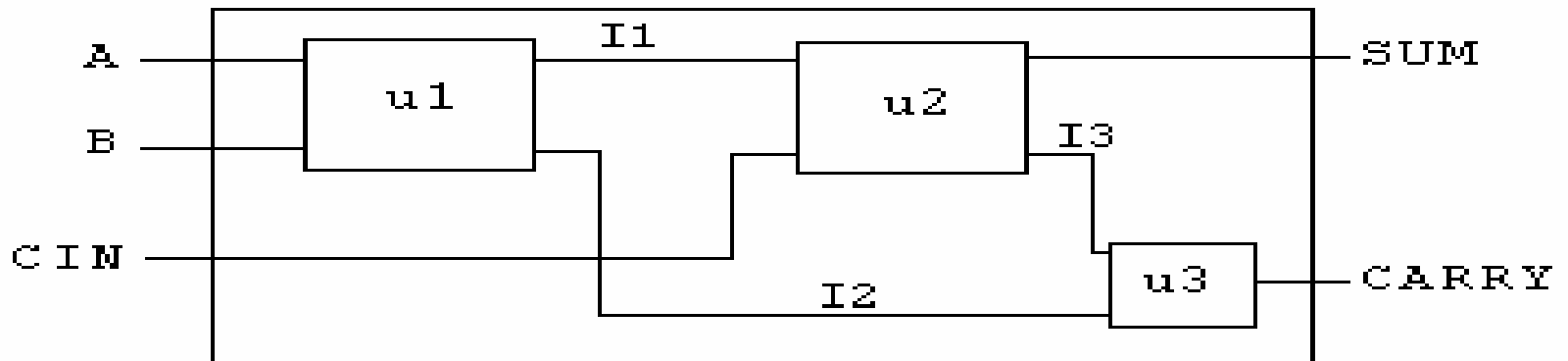
```
entity FULLADD is  
port (A, B, CIN : in bit;  
      SUM, CARRY : out bit);  
end FULLADD;
```



VHDL结构要点

- 实例(FullAdd)-architecture

```
architecture STRUCT of FULLADD is  
    signal I1, I2, I3 : bit;  
    -- other declarations  
begin  
    u1: HALFADD port map (A, B, I1, I2);  
    u2: HALFADD port map (I1, CIN, SUM, I3);  
    u3: ORGATE port map (I3, I2, CARRY);  
end STRUCT;
```



```

entity FULLADD is
port (A, B, CIN : in bit;
      SUM, CARRY : out bit);
end FULLADD;

architecture STRUCT of FULLADD is
  signal I1, I2, I3 : bit;
  component HALFADD
    port (A,B : in bit;
          SUM, CARRY : out bit);
  end component;
  component ORGATE
    port (A,B : in bit;
          Z : out bit);
  end component;
begin
  u1:HALFADD port map (A,B,I1,I2);
  u2:HALFADD port map (I1,CIN,SUM,I3);
  u3:ORGATE port map (I3,I2,CARRY);
end STRUCT;

```

VHDL中的设计单元

除了entity(实体)和architecture(构造体)外还有另外三个可以独立进行编译的设计单元

- **Package**（包集合）属于库结构的一个层次，存放信号定义、常数定义、数据类型、元件语句、函数定义和过程定义。
- **Package Body** 具有独立对端口(port)的package
- **configuration**（配置）描述层与层之间的连接关系以及实体与构造体之间关系。

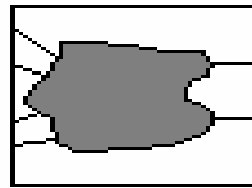
VHDL中的设计单元

- VHDL中的设计单元（可以独立编译）

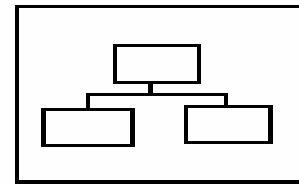
Design units...



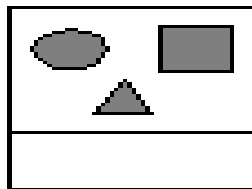
Entity



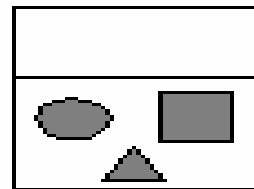
architecture



configuration



Package



Package body

Library 库的概念

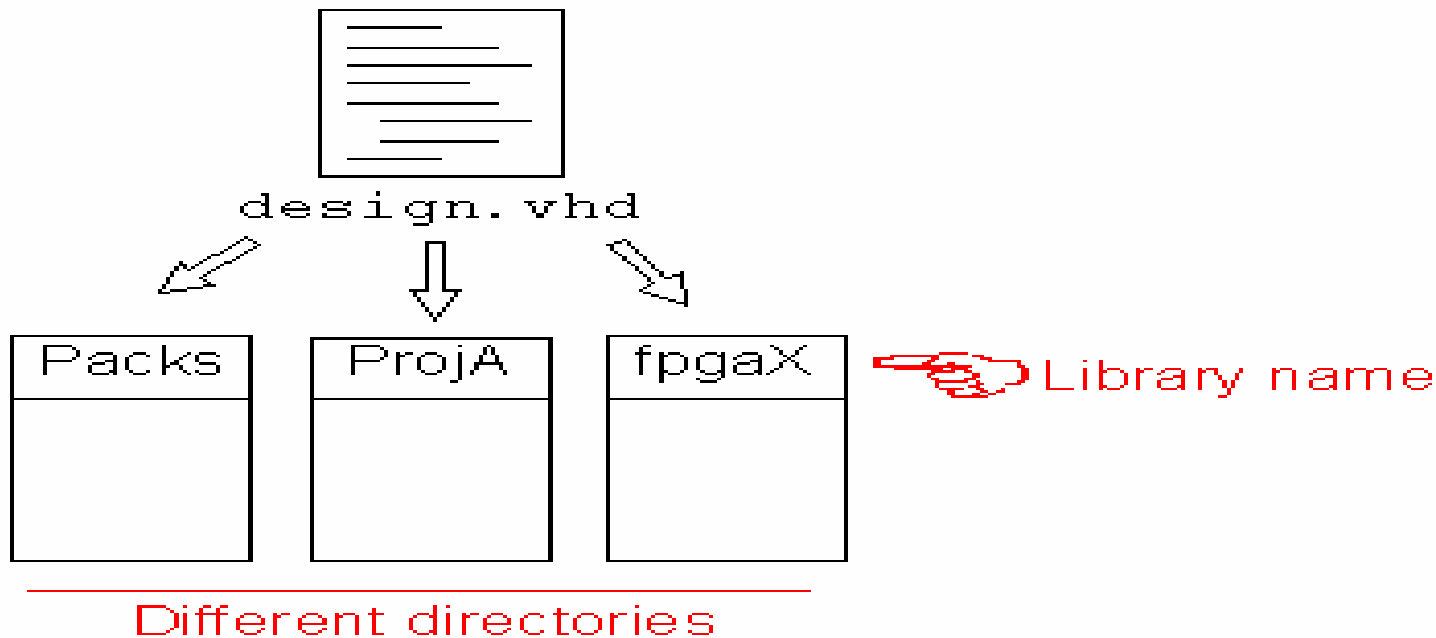
库： 数据的集合。内含各类包定义、实体、构造体等

- STD库 --VHDL的标准库
- IEEE库 -- VHDL的标准库的扩展
- 面向ASIC的库 --不同的工艺
- 不同公司自定义的库
- 普通用户自己的库

Library 库的概念

- 用户自己的库

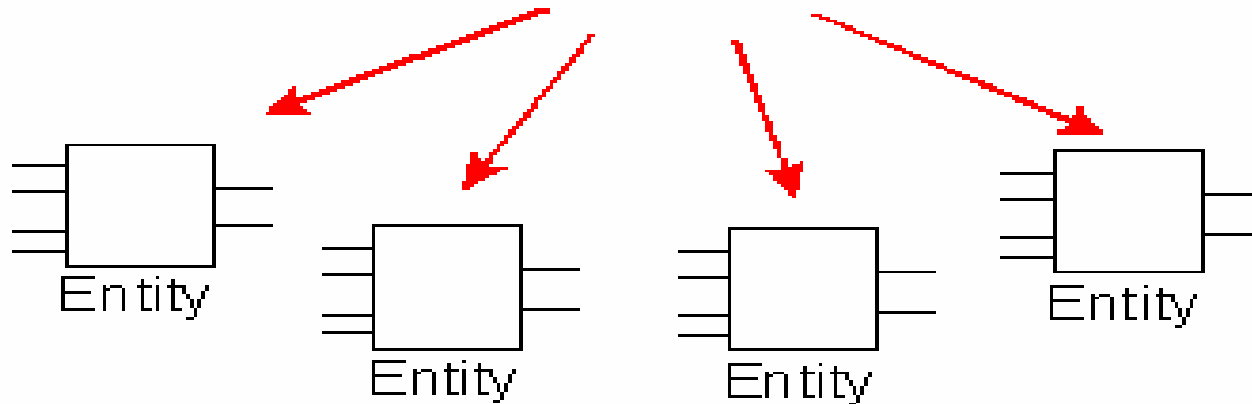
当您的VHDL文件被编译后，编译的结果储存在特定的目录下，这个目录的逻辑名称即Library，此目录下的内容亦即是这个Library的内容。



Package 包的概念

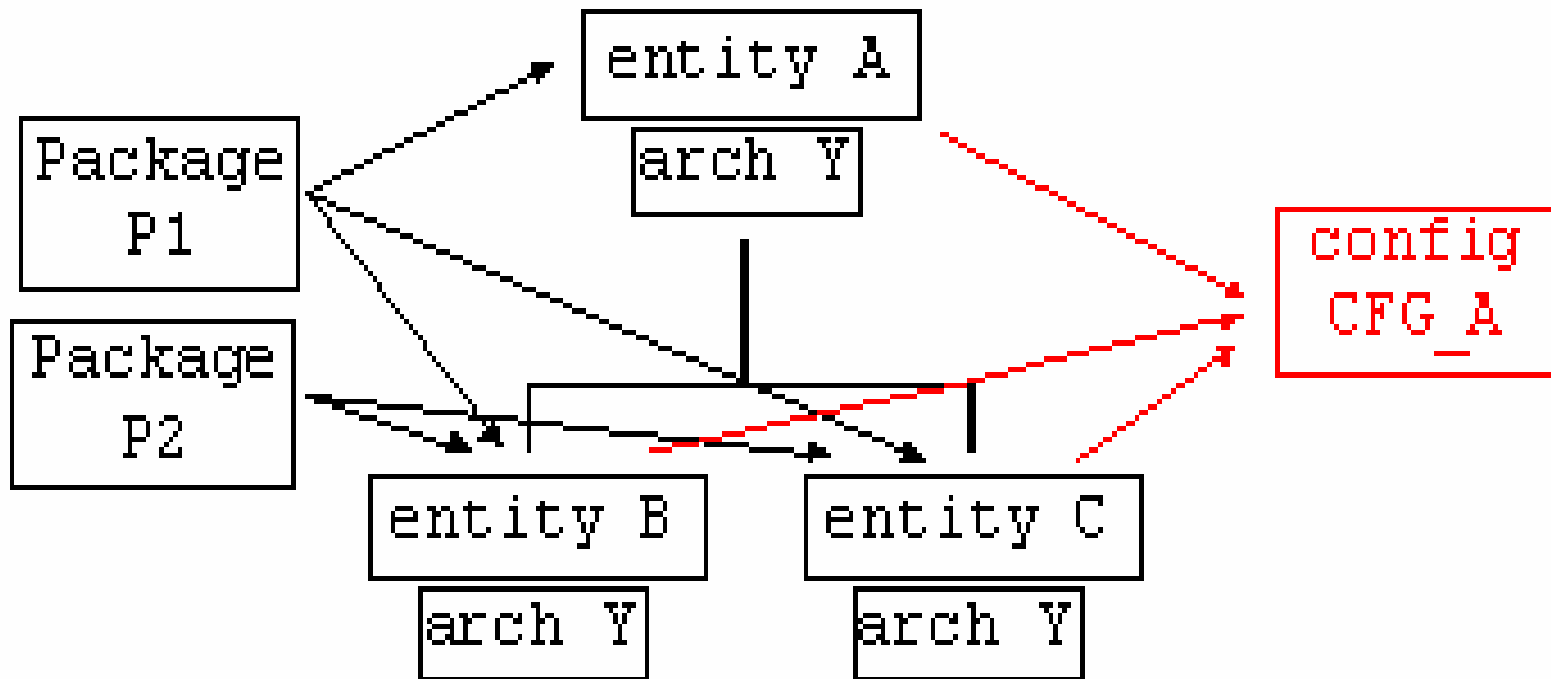
- Package (包)

```
package PROJECT_X is  
    -- definitions  
end PROJECT_X;
```



VHDL中的结构关系

- 结构关系



VHDL简介及其结构

- 本讲结束
- 下一讲:
VHDL中的对象、操作符、数据类型

第二讲、VHDL对象、操作符、数据类型

- 通过本课的学习您可以了解以下几点
 - 1、VHDL的基本类型
 - 2、如何在VHDL中定义类型
 - 3、VHDL的信号定义
 - 4、如何在VHDL中对信号赋值
 - 5、VHDL中的操作符

VHDL对象、操作符、数据类型

- 对象object
对客观实体的抽象和概括
- VHDL中的对象有：

- 1、Constant（常量）在程序中不可以被赋值
- 2、Variable（变量）在程序中可以被赋值(用“:=”), 赋值后立即变化为新值。
- 3、Signal（信号）在程序中可以被赋值(用“<="), 但不立即更新，当进程挂起后，才开始更新。

VHDL对象、操作符、数据类型

- VHDL中的对象使用：

variable

x, y: integer; --定义了整数型的变量对象x, y

constant

Vcc: real; --定义了实数型的常量对象Vcc

signal

clk, reset: bit; --定义了位类型的信号对象clk, reset

VHDL中的对象使用

- 注意

- 1、variable **只能定义**在process和subprogram（包括function和procedure）中，不可定义在其外部。
- 2、signal **不能定义**在process和subprogram（包括function和procedure）中，只可定义在其外部。

VHDL对象、操作符、数据类型

- 对象的属性

类似于其它面向对象的编程语言如VB、VC、DELPHI

用法格式: 对象' 属性

例子: clk' event --表明信号clk的event属性

常用的属性:

Signal 对象的常用属性有:

event : 返回boolean值, 信号发生变化时返回true

last_value: 返回信号发生此次变化前的值

last_event: 返回上一次信号发生变化到现在变化的间隔时间

VHDL对象、操作符、数据类型

- Signal 对象的常用属性有：接上页

`delayed[(时延值)]`: 使信号产生固定时间的延时并返回
`stable[(时延值)]`: 返回boolean, 信号在规定时间内没有变化返回true
`transaction`: 返回bit类型, 信号每发生一次变化, 返回值翻转一次

例子:

```
A<=B'delayed(10 ns);    --B延时10ns后赋给A;  
if (B'Stable(10 ns)) ;    --判断B在10ns中是否发生变化
```

VHDL对象、操作符、数据类型

- 属性应用

信号的event和last_value属性经常用来确定信号的边沿

例如：

判断clk的上升沿

```
if ( (clk'event)and (clk='1') and(clk'last_value='0')) then
```

判断clk的下降沿

```
if ( (clk'event)and (clk='0') and(clk'last_value='1')) then
```

VHDL 的基本类型

- 1、**bit**(位): `0` 和 `1`
- 2、**bit-Vector**(位矢量): 例如: ``00110``
- 3、**Boolean** “ture”和“false”
- 4、**time** 例如: 1 us、100 ms, 3 s
- 5、**character** 例如: ‘a’、’n’、’1’、’0’
- 6、**string** 例如: “sdfsd”、” my design”
- 7、**integer** 32位例如: 1、234、-2134234
- 8、**real** 范围-1.0E38~+1.0E38

例如: 1.0、2.834、3.14、0.0

VHDL 的基本类型

9、**natural** 自然数 和 **positive** 正整数

10、**severity level** （常和assert语句配合使用）

包含有：note、warning、error、failure

- 以上十种类型是VHDL中的标准类型，在编程中可以直接使用。使用这十种以外的类型，需要自行定义或指明所引用的Library(库)和Package(包)集合

VHDL 的基本类型

- 例子一

Legal declarations:

```
signal Z_BUS:bit_vector(3 downto 0);  
signal C_BUS:bit_vector(1 to 4);
```

Illegal declarations:

```
signal Z_BUS:bit_vector(0 downto 3);  
signal C_BUS:bit_vector(3 to 0);
```

VHDL 的基本类型和赋值

- 例子二

```
signal A, B, Z : bit;  
signal X_INT : integer;
```

signal assignment
statement



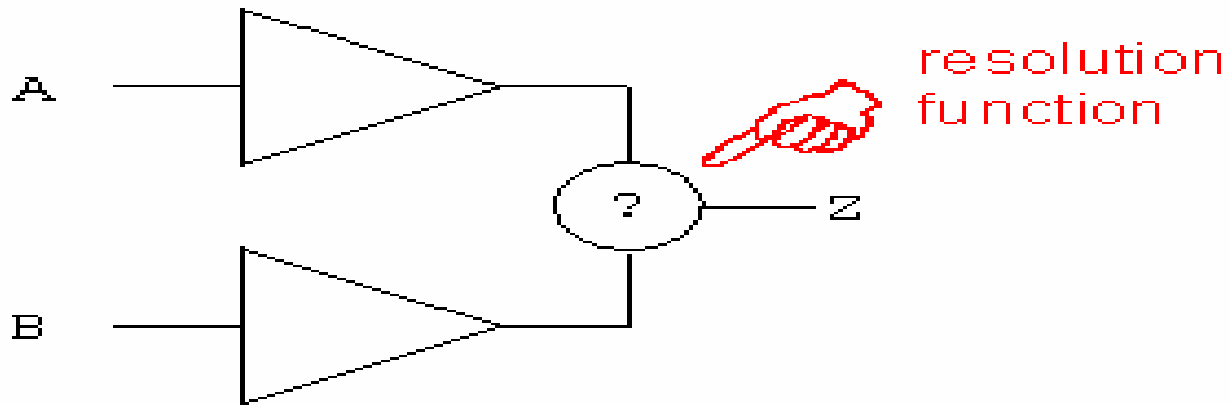
```
Z <= A;
```

VHDL 的基本类型和赋值

- 例子三

```
signal A,B,Z : bit;  
signal X_INT : integer;
```

```
Z <= A;  
Z <= B;
```



例子中信号Z有两个驱动A和B；Z必须定义为一种新的数据类型，否则Z将无法决定取值，语句视为非法。

VHDL 的基本类型和赋值

- 例子四

```
signal Z_BUS:bit_vector(3 downto 0);  
signal C_BUS:bit_vector(1 to 4);
```

```
Z_BUS <= C_BUS;
```

Is the same as:

```
Z_BUS(3) <= C_BUS(1);  
Z_BUS(2) <= C_BUS(2);  
Z_BUS(1) <= C_BUS(3);  
Z_BUS(0) <= C_BUS(4);
```

Assignment is by position!

VHDL 的基本类型和赋值

- 例子五

```
signal Z_BUS:bit_vector(3 downto 0);  
signal C_BUS:bit_vector(1 to 4);
```

Legal:

```
Z_BUS(3 downto 2) <= "00";  
C_BUS(2 to 4) <= Z_BUS(3 downto 1);
```

Illegal:

```
Z_BUS(0 to 1) <= "11";
```

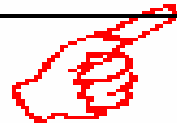
- 要点：赋值语句中的方向应和声明中的方向一样

VHDL 的基本类型和赋值

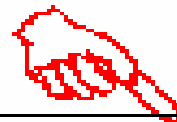
- 连接操作符---使用&

```
signal Z_BUS : bit_vector(3 downto 0);  
signal A, B, C, D : bit;  
signal BYTE : bit_vector(7 downto 0);  
signal A_BUS : bit_vector(3 downto 0);  
signal B_BUS : bit_vector(3 downto 0);
```

```
Z_BUS <= A & B & C & D;
```



concatenation
operator



```
BYTE <= A_BUS & B_BUS;
```

VHDL 的基本类型和赋值

- 集合操作---使用 ()

```
signal Z_BUS : bit_vector(3 downto 0);  
signal A, B, C, D : bit;  
signal BYTE : bit_vector(7 downto 0);
```

```
Z_BUS <= (A, B, C, D);
```

aggregate



equivalent to:

```
Z_BUS(3) <= A;  
Z_BUS(2) <= B;  
Z_BUS(1) <= C;  
Z_BUS(0) <= D;
```

VHDL 的基本类型和赋值

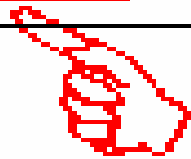
- 集合操作---采用序号

```
signal X : bit_vector(3 downto 0);
```

```
signal A, B, C, D : bit;
```

```
signal BYTE : bit_vector(7 downto 0);
```

```
X <= (3=>'1', 1 downto 0=>'1', 2 => B);
```



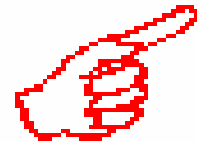
assignment by name

VHDL 的基本类型和赋值

- 集合操作--采用others

```
signal X : bit_vector(3 downto 0);  
signal A, B, C, D : bit;  
signal BYTE : bit_vector(7 downto 0);
```

```
X <= (3=>'1', 1=>'0', others => B);
```



在VHDL中定义自己的类型

- 通用格式

TYPE 类型名 IS 数据类型定义

- 用户可以定义的数据类型

枚举类型enumerated、*整数型integer*、

实数型real、*数组类型array*、

纪录类型record、*时间类型time*、

文件类型file、*存取类型access*

在VHDL中定义自己的类型

- 枚举类型enumerated

- 格式

type 数据类型名 *is* (元素, 元素.....) ;

- 例子

```
type week is (sun,mon,tue,thu,fri,sat);
```

```
type std_logic is ('1','0','x','z');
```

在VHDL中定义自己的类型

- 整数类integer和实数类real

- 格式

type 数据类型名 **is** 数据类型定义 约束范围;

- 例子

```
type week is integer range 1 to 7;
```

```
type current is real range -1E4 to 1E4
```

在VHDL中定义自己的类型

- 数组类型array

- 格式

type 数据类型名 *is array* 范围 *of* 元数据类型名

- 例子

type week *is array* (1 to 7) *of* integer;

type deweek *is array* (1 to 7) *of* week;

在VHDL中定义自己的类型

- 时间类型time
- 格式

type 数据类型名 *is* 范围

units 基本单位;

单位;

end units

在VHDL中定义自己的类型

- 时间类型例子

```
type time is range -1E18 to 1E18
units
  us;
  ms=1000 us;
  sec=1000 ms;
  min=60 sec;
end units
```

注意: 引用时间时, 有的编译器要求**量**与**单位**之间应有一个空格如: 1 ns; 不能写为1ns;

在VHDL中定义自己的类型

- 纪录类型record
- 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;

在VHDL中定义自己的类型

- 纪录类型的例子

```
type order is record
```

```
  id:integer;
```

```
  date:string;
```

```
  security:boolean;
```

```
end record;
```

- 引用:signal flag:boolean;

```
  signal order1:order;
```

```
  order1<=(3423,"1999/07/07",true);
```

```
  flag<=order1.security;
```

IEEE 1164中定义的类型

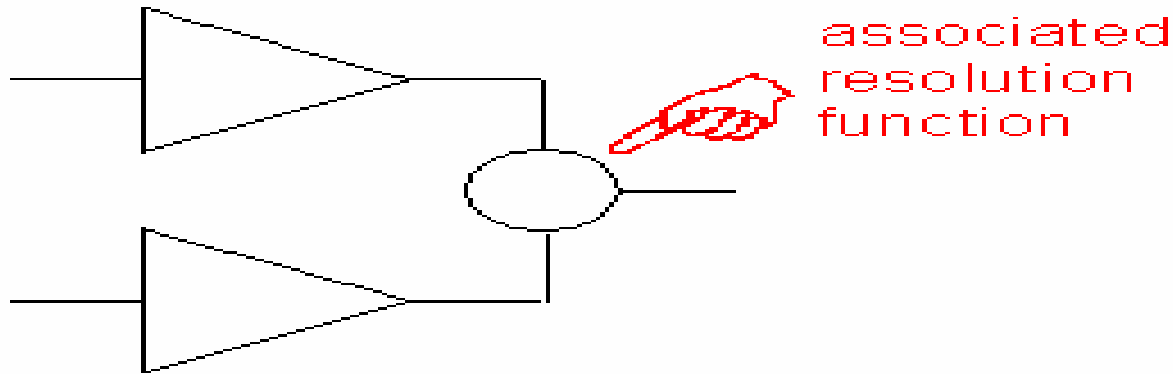
- `std_ulogic` 是对位(bit)类型的扩展,只允许一个驱动源

```
type std_ulogic is (  
    'U' ,    uninitialised  
    'X' ,    unknown  
    '0' ,    logic 0      | Strong drive  
    '1' ,    logic 1  
    'Z' ,    High impedance  
    'W' ,    unknown      | Weak drive  
    'L' ,    logic 0      | (not often used)  
    'H' ,    logic 1  
    '-' ,    Don't care
```

IEEE 1164中定义的类型

- Std_logic同std_ulogic 一样有九个状态,允许一个或多个驱动源

std_logic



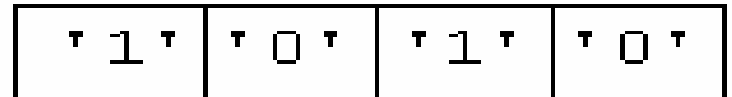
One or more drivers

IEEE 1164中定义的类型


- Std_unlogic_vector和std_logic_vector

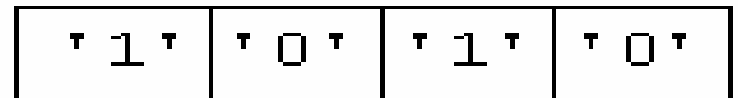
`std_ulogic_vector`

array of std_ulogic 



`std_logic_vector`

array of std_logic 



IEEE 1164中定义的类型

Std_unlogic、std_ulogic_vector

std_logic_vector和std_unlogic_vector 类型

均定义在package(包) standard_logic_1164中

在使用这四种类型时应加以说明,

例如:library ieee;

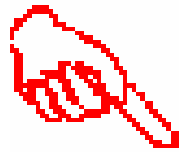
```
use ieee.std_logic_1164.all;
```

注: standard_logic_1164位于IEEE库中

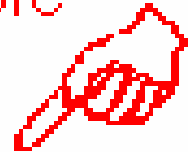
类型使用例子

- 例子一(声明使用的库和包)

makes library visible



makes all contents of package visible

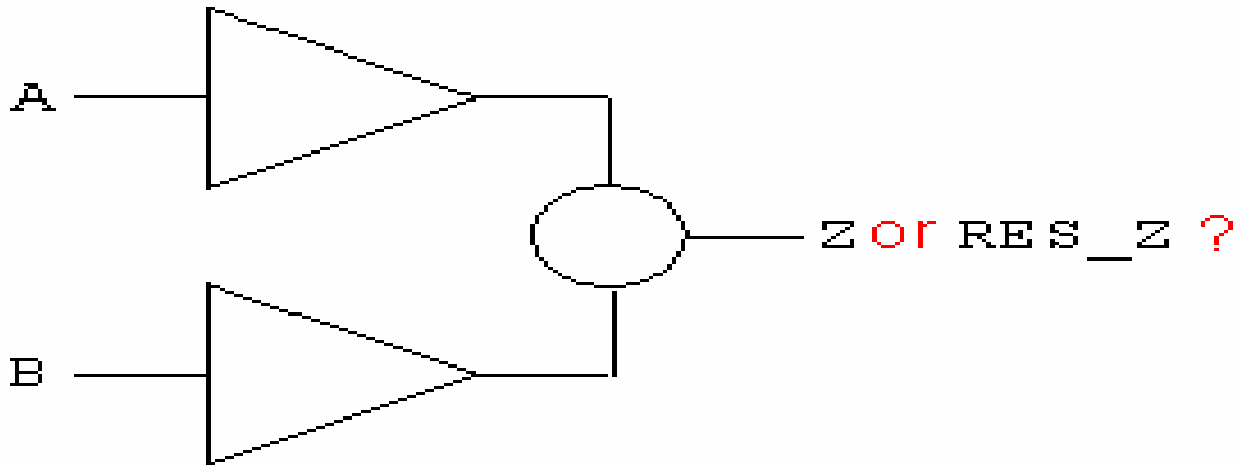


```
library IEEE;  
use IEEE.Std_Logic_1164.all;  
  
entity MVLS is  
    port (A, B, : in std_ulogic;  
          Z      : out std_ulogic);  
end MVLS ;
```

类型使用例子

- 例子二 std_ulogic 和std_logic的区别

```
signal A, B, Z : std_ulogic;  
signal RES_Z : std_logic;
```



```
Z <= A;  
Z <= B;
```



```
RES_Z <= A;  
RES_Z <= B;
```



类型使用例子

- 练习一: 下面那一个是正确的

```
signal A_BUS : bit_vector (3 downto 0);  
signal Z_BUS : bit_vector (3 downto 0);  
signal A_BIT, B_BIT, C_BIT, D_BIT : bit;  
signal BYTE : bit_vector (7 downto 0);
```

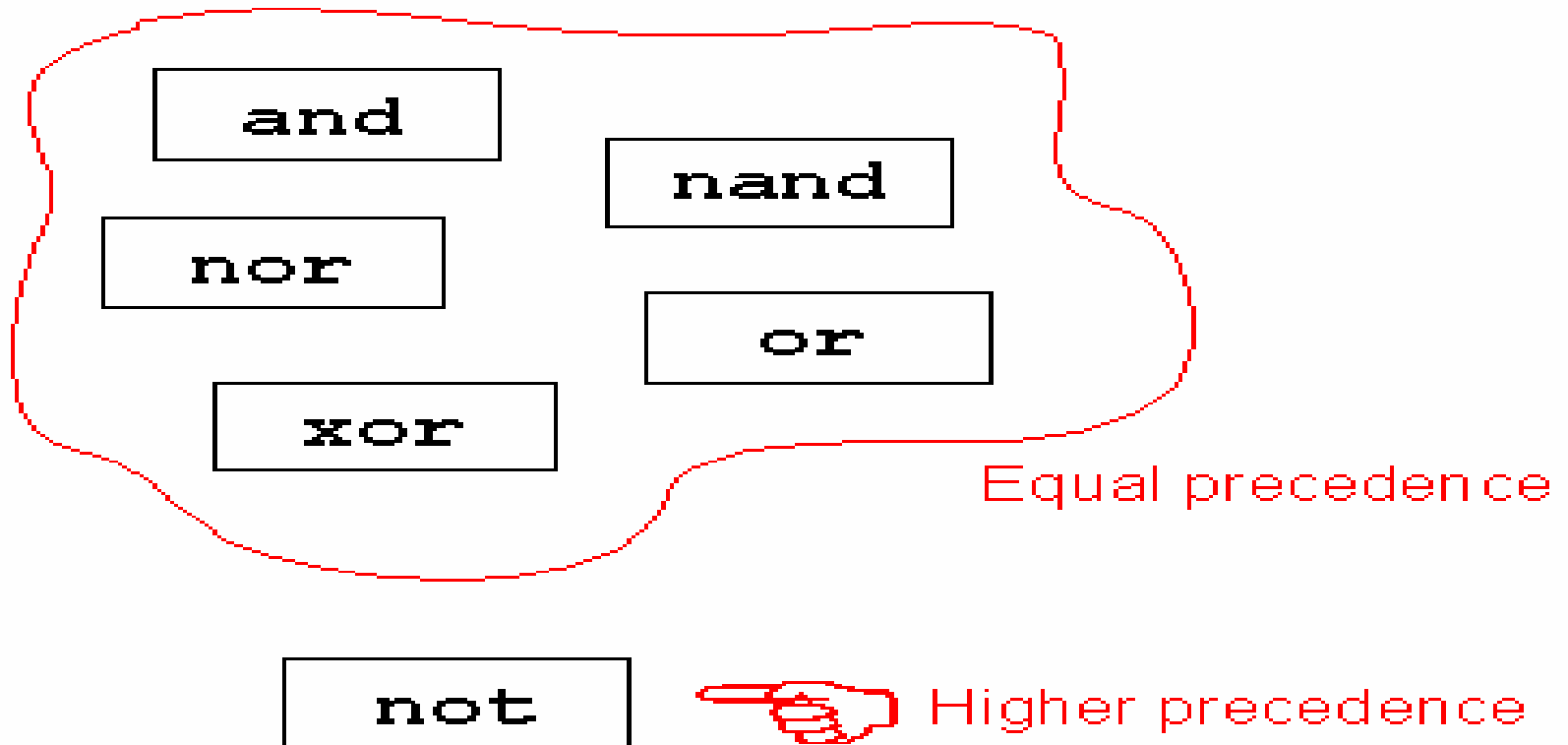
1. `BYTE <= (OTHERS => '1');`
2. `Z_BUS <= A_BIT & B_BIT;`
3. `A_BUS <= ('1', B_BIT, '0', D_BIT);`
4. `A_BUS (0 to 1) <= (OTHERS => '0');`

VHDL中的操作符

- 分类
 - 1、逻辑操作符
 - 2、关系操作符
 - 3、数学运算符

VHDL中的操作符

1、逻辑操作符有：



VHDL中的操作符

- 逻辑操作符的应用类型

`bit`

`boolean`

`bit_vector`

`std_logic_vector`

`std_logic`

`std_ulogic_vector`

`std_ulogic`

VHDL中的操作符

- 逻辑操作符的应用例子

```
signal A_BUS, B_BUS, Z_BUS :  
        std_ulogic_vector (3 downto 0);
```

```
Z_BUS <= A_BUS and B_BUS;
```

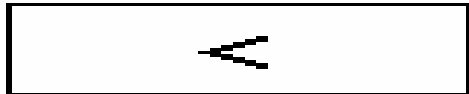


equivalent to:

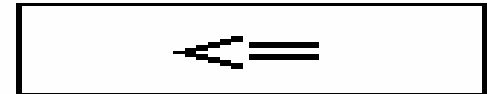
```
Z_BUS (3) <= A_BUS (3) and B_BUS (3);  
Z_BUS (2) <= A_BUS (2) and B_BUS (2);  
Z_BUS (1) <= A_BUS (1) and B_BUS (1);  
Z_BUS (0) <= A_BUS (0) and B_BUS (0);
```

VHDL中的操作符

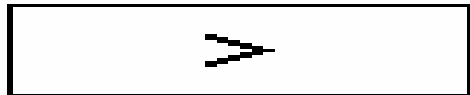
2、关系运算符有



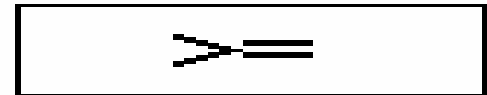
less than



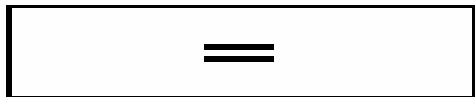
less than
or equal to



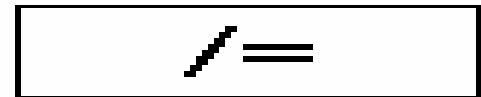
greater than



greater than
or equal to



equal to



not equal
to

VHDL中的操作符

- 关系运算符的应用

Operands of same type

arrays of different lengths:

Align left 

 Compare to right

1	0	1	1
---	---	---	---

1	1	1
---	---	---

Greatest!

- ARRAY（数组）没有数字概念，数组“111”不等于7

VHDL中的操作符

3、数学运算符

+

addition

-

subtraction

multiplication

/

division

exponential

abs

absolute
value

mod

modulus

rem

remainder

注意:上述运算符应用于 integer, real, time 类型, 不能用于 vector(如果希望用于 vector, 可以使用库 IEEE 的 std_logic_unsigned 包, 它对算术运算符进行了扩展)

VHDL中的操作符

- VHDL中的操作符应用要点
 - 1、VHDL属于强类型，不同类型之间不能进行运算和赋值，可以进行数据类型转换
 - 2、vector不表示number
 - 3、array 不表示number

VHDL中的操作符

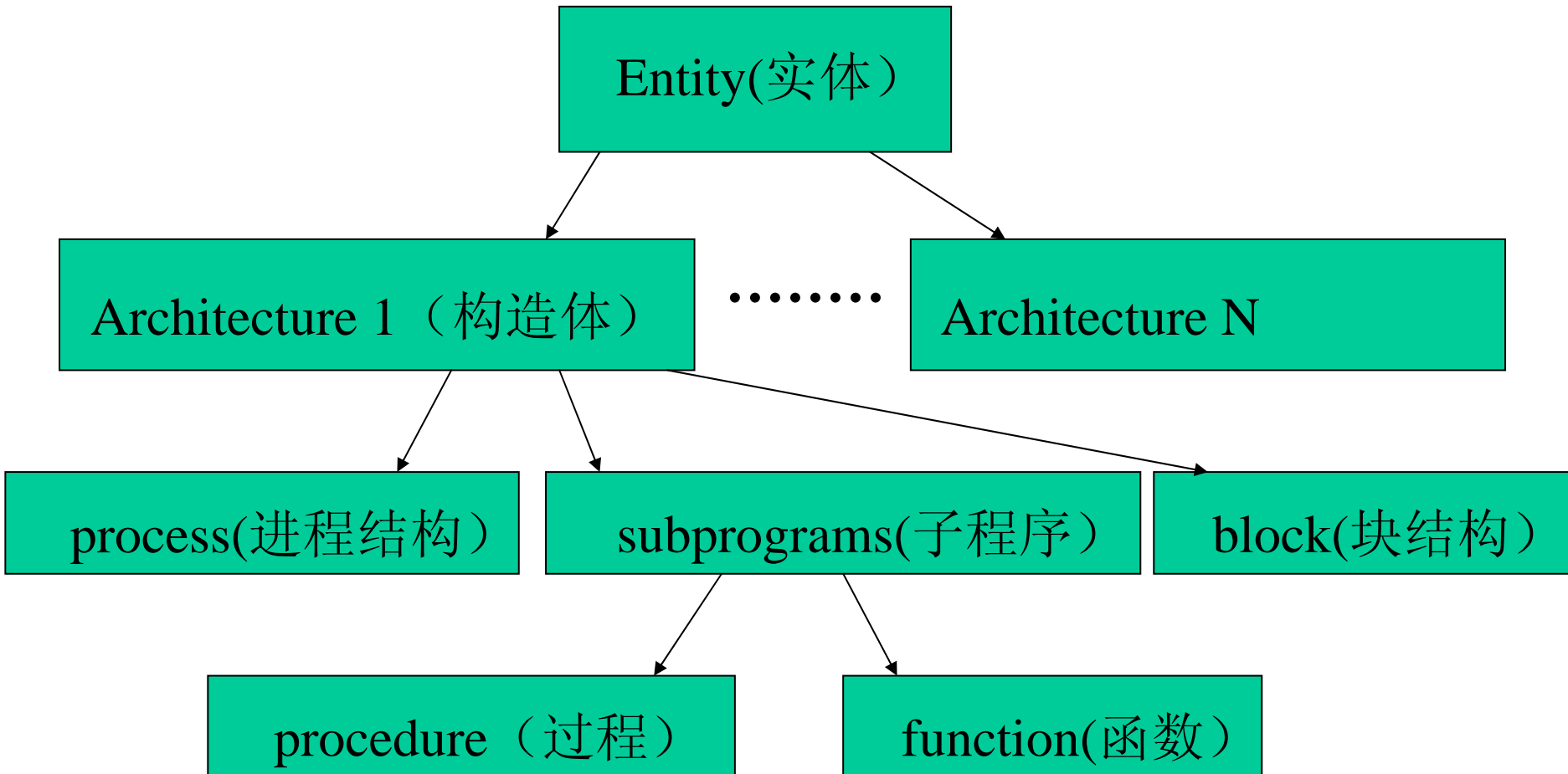
- 本讲结束
- 下一讲:
VHDL中的控制语句及模块

第三讲VHDL中的控制语句及模块

- 通过本讲您将会学到
 - 1、Block的编写
 - 2、Process的编写
 - 3、function 和 procedure的编写
 - 4、VHDL中的流程控制语句的书写

VHDL中的控制语句及模块

- 回顾第一讲的内容



VHDL中的控制语句及模块

- 基本概念

- 1、并行处理（concurrent）

语句的执行与书写顺序无关，并行块内的语句时同时执行的

- 2、顺序处理（sequential）

语句的执行按书写的先后次序，从前到后顺序执行。这种方式和其他普通编程语言(如c， pascal)是一样的。

VHDL中的控制语句及模块

- Architecture 中的语句及子模块之间是并行处理的
- 子模块block中的语句是并行处理的
- 子模块process中的语句是顺序处理的
- 子模块subprogram中的function和procedure是顺序处理的

VHDL中的architecture

Architecture（构造体）的格式为：（第一讲）

Architecture 构造体名 **of** 实体名 **is**

[定义语句] 内部信号、常数、元件、数据类型、函数等的定义

begin

[并行处理语句和block、process、function、procedure]

end 构造体名；

Architecture中的Block

- Block
- 格式

块名:

BLOCK

[定义语句]

begin

[并行处理语句concurrent statement]

end block 块名

Architecture中的Block

- 条件Block
- 格式

块名:

BLOCK [(布尔表达式)]

[定义语句]

begin

[并行处理语句concurrent statement

[信号]<= ***guarded*** [信号,延时];

end block 块名

Architecture中的Block

- Block 例子

```
myblock1:
```

```
  block (clk= '1' )
```

```
    signal: qin: bit:  
    = '0' ;
```

```
  begin
```

```
    qout<= guarded qin ;
```

```
  end block myblock1
```

```
myblock2:
```

```
  block
```

```
    begin
```

```
      qout<=qin;
```

```
    end block myblock2
```

Architecture中的process

- Process
- 格式

[进程名:]

process [(触发信号列表)]

[定义语句;]

begin

[串行处理语句sequential statement;]

end process

Architecture中的process

- process例子

exp1:

```
process (clk, qin)
  variable: qin: bit: = '0' ;
begin
  qout<=qin;
end process
```

exp2:

```
process
  begin
    wait on clk,qin;
    qout<=qin;
  end process
```

- process例子-值的更新

```
architecture SEQUENTIAL of MULTIPLE is
    signal Z, A, B, C, D : std_ulogic;
begin
    process (A, B, C, D)
    begin
        Z <= A and B;
        Z <= C and D;
    end process; ← Z updated after
end SEQUENTIAL;                               process suspends
```

- 分析:

当A、B、C、D中任一信号发生变化时，进程将开始执行，当执行 $Z \leq A \text{ and } B$ 后，Z 的值不会立即变化；同理执行 $Z \leq C \text{ and } D$ 后Z 的值也不会立即变化。当执行end process后，Z 的值才开始更新，同时系统挂起开始等待敏感信号。

Architecture中的process

- Process中敏感信号列表的普遍原则是：

在process中，其值被引用的信号应当出现在敏感信号列表中

```
MUX: process (A, B, SEL)
begin
    if SEL = '1' then
        Z <= A;
    else
        Z <= B;
    end if;
end process MUX;
```

例子;

二选一的选择器:

A、B为输入信

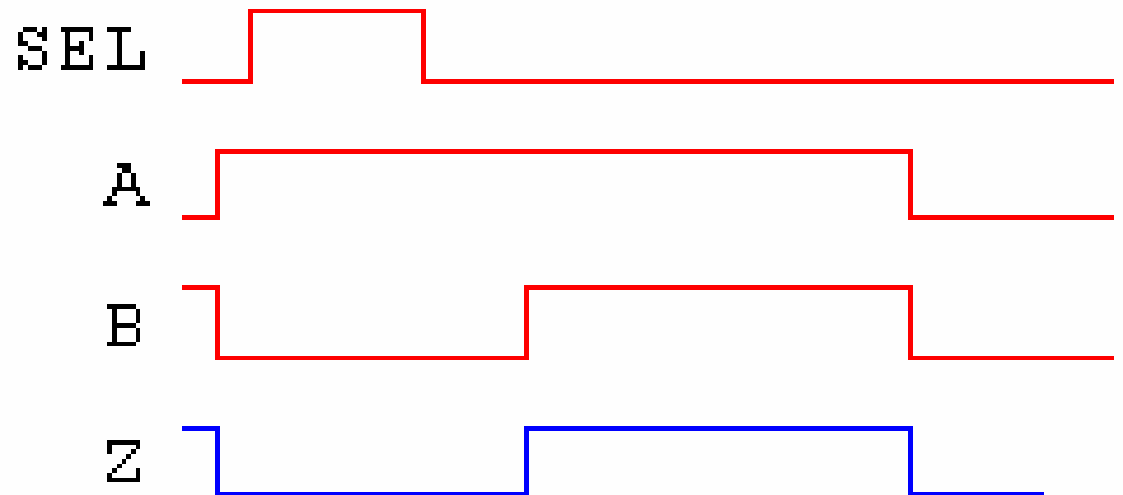
号; SEL为选路信

号; Z为输出信

号;

- 不符和设计要求

```
MUX: process (A, B)
begin
    if SEL = '1' then
        Z <= A;
    else
        Z <= B;
    end if;
end process MUX;
```



Architecture中的subprogram

- Function（函数）
- 格式：

function 函数名（参数1，参数2）

[定义语句]

return 数据类型名 ***is*** [定义语句]

begin

[顺序执行语句]

return [返回变量名]

end 函数名

Architecture中的subprogram

- Function 例子

```
function max (a, b: bit)
  return boolean is variable : flag:
  boolean;
begin
  if (a=b) then
    flag<=true;
  end if
  return flag;
end max
```

Function 中的参数不用说明方向(因为只有一种方向in)

Architecture中的subprogram

- procedure（过程）
- 格式：

procedure 过程名（参数1，参数2） ***is***

[定义语句]

begin

[顺序执行语句]

end 过程名

Architecture中的subprogram

- Procedure例子

```
procedure max (a, b: in bit;  
              flag: out boolean) is  
begin  
  if (a=b) then  
    flag<=true;  
  end if  
end max;
```

顺序执行语句sequential statement

- Wait语句
- assert语句
- If 语句
- case语句
- for loop语句
- while 语句

顺序执行语句sequential statement

- Wait语句
- 书写格式

```
wait; --无限等待
wait on [信号列表] --等待信号变化
wait until [条件]; --等待条件满足
wait for [时间值]; --等待时间到
```

- 功能

wait语句使系统暂时挂起 (等同于end process), 此时, 信号值开始更新。条件满足后, 系统将继续运行。

顺序执行语句sequential statement

- Wait语句例子

```
process (a, b)
begin
    y<=a and b;
end process
```

等同于

```
process
begin
    wait on a,b;
    y<=a and b;
end process
```

```
process (a, b)
begin
    wait on a, b;
    y<=a and b;
end process
```

错误 如果process中已有敏感信号
进程中不能使用wait 语句

顺序执行语句sequential statement

- Wait语句例子

如果process中没有敏感信号列表，其进程中也没有wait语句，则process中的程序代码循环执行

```
process
  begin
    clk<=not clk after 50 ns ;
  end process
```

功能: 产生频率为100 ns的clk信号

顺序执行语句sequential statement

- Assert语句格式

```
assert 条件 [report 输出信息] [severity]
```

说明: 条件为true 时执行下一条语句, 为false 时输出错误信息和错误的严重级别

- 例子

```
....  
assert (sum=100) report “sum /=100” severity error;  
next statement  
.....
```

顺序执行语句sequential statement


- If 语句格式

```
if 条件 then
    [顺序执行语句]
[else]
    [顺序执行语句]
end if
```

```
if 条件 then
    [顺序执行语句]
[elsif]
    [顺序执行语句]
[elsif]
    [顺序执行语句]
.....
[else]
end if
```

顺序执行语句sequential statement

- If 语句例子

```
process (A, B, C, X)
begin
    Always
    executes if  x = "0000"
    if (X = "0000") then
        Z <= A;
    elsif (X <= "0101") then
        Z <= B;
    else
        Z <= C;
    end if;
end process;
```

顺序执行语句sequential statement

- Case 语句格式

Case 表达式 is

when 条件表达式=> 顺序处理语句

when 条件表达式=> 顺序处理语句

.....

when others=> 顺序处理语句

end case

原则: 1、*完全性*:表达式所有可能的值都必须说明, 可以用 others

2、*唯一性*:相同表达式的值只能说明一次

顺序执行语句sequential statement

- Case 语句例子,条件表达式可以有多种形式

```
process (A, B, C, X)
begin
```

```
  case X is
```

```
    when 0 to 4 =>
```

```
      Z <= B;
```

```
    when 5 =>
```

```
      Z <= C;
```

```
    when 7 | 9 =>
```

```
      Z <= A;
```

```
    when others =>
```

```
      Z <= 0;
```

```
  end case;
```

```
end process;
```




顺序执行语句sequential statement

- Case 语句例子

```
process (A, B, C, X)
begin
  case X is
    when 0 to 4 =>
      Z <= B;
    when 3 =>
      Z <= C;
    when 7 | 9 =>
      Z <= A;
    when others =>
      Z <= 0;
  end case;
end process;
```

 only specify
once

 cover all
possible
values

顺序执行语句sequential statement

- For loop 语句格式

```
For 循环变量 in 范围 loop  
[顺序处理语句]  
end loop
```

- For loop 语句例子

```
For i in 1 to 10 loop  
    sum=sum+1;  
end loop
```

注意: 循环变量不需要定义(声明);例子中 i 不需要定义

顺序执行语句sequential statement

- 在loop语句中可以用next来跳出本次循环，也可以用exit来结束整个循环状态

next 格式: next [标号] [when 条件];

exit 格式: exit [标号] [when 条件];

```
For i in 1 to 10 loop
  sum=sum+1;
  next when
sum=100;
end loop
```

```
For i in 1 to 10 loop
  sum=sum+1;
  exit when
sum=100;
end loop
```

顺序执行语句 sequential statement

- While 语句格式

```
while 条件 loop  
[顺序处理语句]  
end loop
```

- While 语句例子

```
While i<10 loop  
sum=sum+1;  
i=i+1;  
end loop
```

并行处理语句 concurrent statement

- 1、信号赋值操作
- 2、带条件的信号赋值语句
- 3、带选择的信号赋值语句

并行处理语句concurrent statement

- 信号赋值操作
- 符号“<=”进行信号赋值操作的，
- 它可以用在顺序执行语句中，
- 也可以用在并行处理语句中
- **注意**

1、用在并行处理语句中时，符号<=右边的值是该语句的敏感信号，即符号<=右边的值发生变化就会重新激发此条赋值语句，也即符号<=右边的值不变化时，此条赋值语句就不会执行。如果符号<=右边是常数则赋值语句一直执行。

2、用在顺序执行语句中时，没有以上说法。

并行处理语句concurrent statement

- 赋值语句例子

```
Myblock: Block
```

```
begin
```

```
clr<='1' after 10 ns;
```

```
clr<='0' after 20 ns;
```

```
end block myblock
```

程序执行10 ns后clr 为1，又过 10 ns后 0赋给了clr，此时clr 以前的值1并没有清掉，clr将出现不稳定状态

```
process
```

```
begin
```

```
clr<='1' after 10 ns;
```

```
clr<='0' after 20 ns;
```

```
end block myblock
```

程序执行10 ns后clr 为1，又过 20 ns后 clr的值变为0，

并行处理语句concurrent statement

- 条件信号带入语句格式

```
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 表达式4
```

*注意：*最后的Else 项是必须的；满足*完全性和唯一性*

并行处理语句 concurrent statement

- 条件信号带入语句例子

```
Block  
begin  
  
    sel<=b & a;  
  
    q<=ain  when sel="00"  
  
    else bin when sel="01"  
  
    else cin  when sel="10"  
  
    else din when sel="11"  
  
    else xx;  
end block
```

并行处理语句 concurrent statement

- 选择信号带入语句格式

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;

并行处理语句 concurrent statement

- 选择信号带入语句例子

```
Block
begin

  with sel select

    q<=ain  when sel="00",
             bin when sel="01",
             cin  when sel="10",
             din when sel="11"

             xx; when others;

end block
```

顺序执行语句和并行处理语句

- 顺序执行语句和并行处理语句总结
 - 1、顺序执行语句 `wait`、`assert`、`if -else`、`case`、`for-loop`、`while`语句只能用在`process`、`function`和`procedure`中；
 - 2、并行处理语句（条件信号带入和选择信号带入）只能用在`architecture`、`block`中；

其它语句

- Generic语句

entity and2 *is*

```
generic(rise:time:=10 ns);
```

```
port(a,b: in nit ; c:out bit);
```

end and2;

architecture behav *of* and2 *is*

begin

```
c<=(a xor b) after (rise);
```

end behav

```
entity testand2 is  
  port(ain,bin: in nit ; cout:out bit);  
end testand2;  
architecture behav of testand2 is  
  component and2  
    generic(rise:time); port(a,b: in nit ; c:out bit);  
end component;  
begin  
  c<=(a xor b) after (rise);  
  u0:and2 generic map(20 ns) port map(ain,bin,cout);  
end behav
```

一些例子

```
Signal A,B,C, Y,Z,M,N : integer;  
Signal M,N : integer;  
begin  
  process (A,B,C)  
  begin  
    M<=A;  
    N<=B;  
    Z<=M+N;  
    M<=C;  
    Y<=M+N;  
  end process
```

问题: Z和 Y的最终取值是什么?

- 信号值的更新在进程挂起时, (**M+N**)

一些例子

```
signal A, B, C, Y, Z : integer;
begin
  process (A, B, C)
    variable M, N : integer;
  begin
    M := A;
    N := B;
    Z <= M + N;
    M := C;
    Y <= M + N;
  end process;
```



Y gets C + B

- 变量值的更新立即发生

一些例子

- Z 和 Y 最终取什么值;

```
signal A, B, C, Y, Z : integer;
signal M, N           : integer;
begin
  process (A, B, C, M, N)
  begin
    M <= A;
    N <= B;
    Z <= M + N;
    M <= C;
    Y <= M + N;
  end process;
```



Y gets ?
Z gets ?

(C+B) ; M的变化将重新激发进程运行;

结束语

- 祝贺您完成了VHDL基本内容的学习，希望在实践过程中能学到更多！
- 下一讲：
状态机的设计

第四讲、状态机的设计

概念

- 一类十分重要的时序电路
- 许多数字电路的核心部件

状态机概述

- 状态机的结构：
 - A、组合逻辑部分（状态译码器和输出译码器）
 - B、寄存器部分

- 各部分的功能

- 1、状态译码器

确定状态机的下一个状态

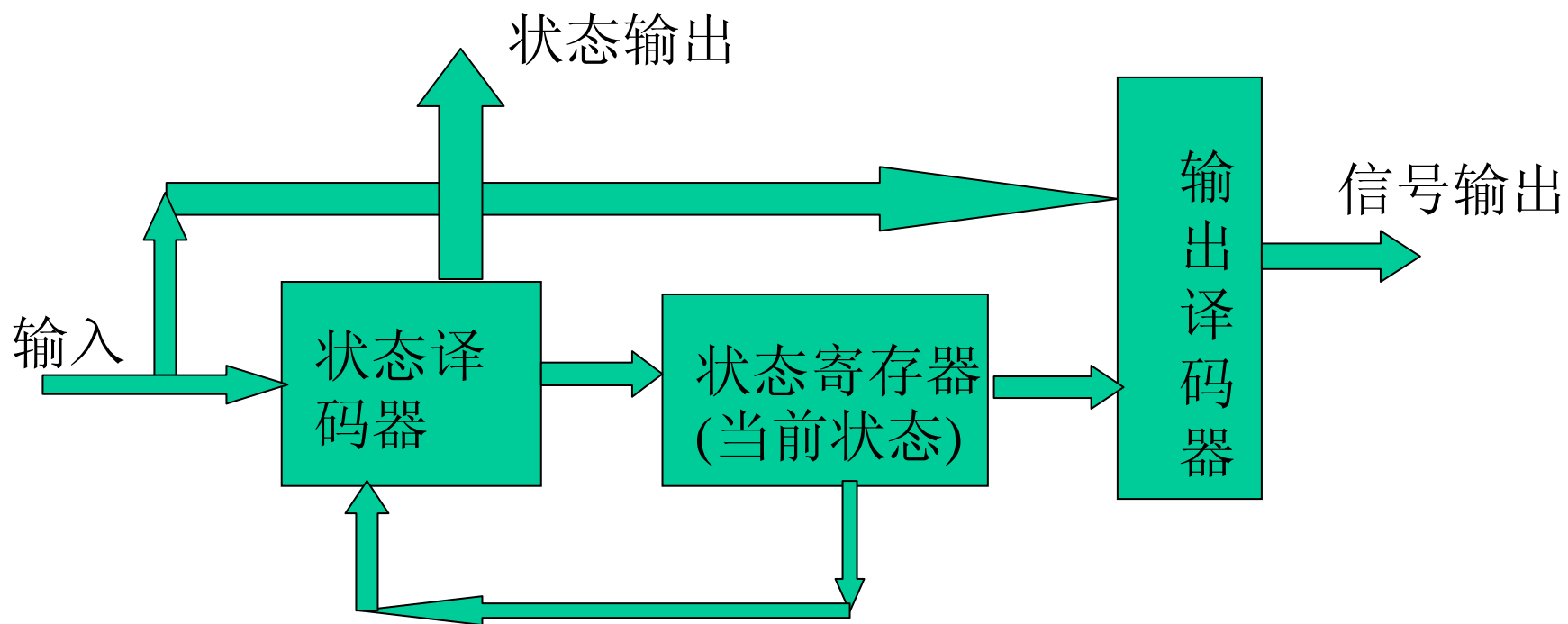
- 2、输出译码器

确定状态机输出

- 3、状态寄存器

存储状态机的内部状态

状态机的结构



状态机的基本操作

1、状态的转换

下一个状态由译码器根据当前状态和输入条件决定。

2、输出信号的产生

输出信号由译码器根据当前状态和输入条件决定

状态机的时序

- 同步时序状态机
由时钟信号触发状态的转换和信号的输出
- 异步时序状态机
状态的转移和输出不与时钟信号同步

注意: 可综合的状态机设计要求使用同步状态机

状态机的设计

- 在产生输出的过程中，由是否使用输入信号可以决定状态机的类型

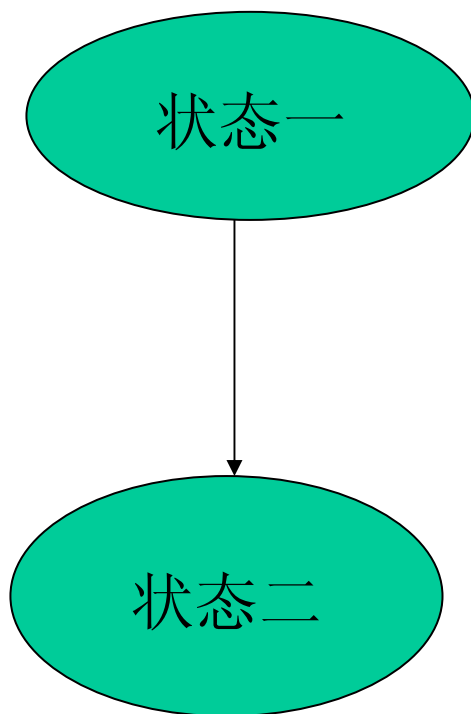
- 两种类型

- 1、米里（mealy）状态机---使用输入信号

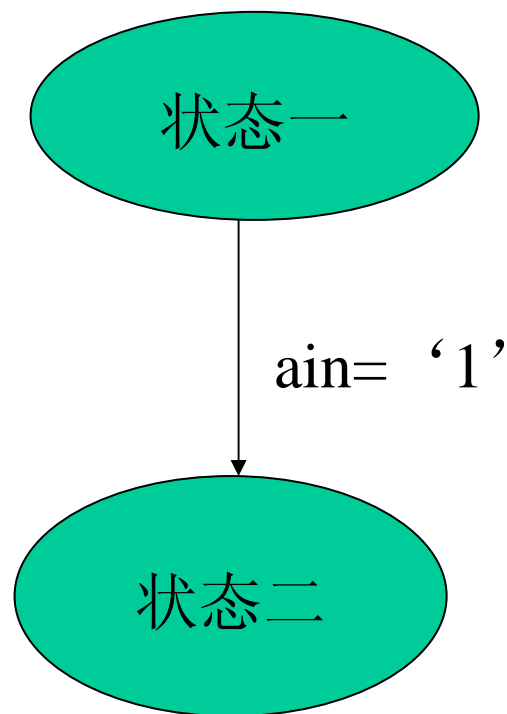
- 2、莫尔（moore）状态机---不使用输入信号

状态机的类型

用状态图表示如下：



莫尔moore状态机



米里mealy状态机

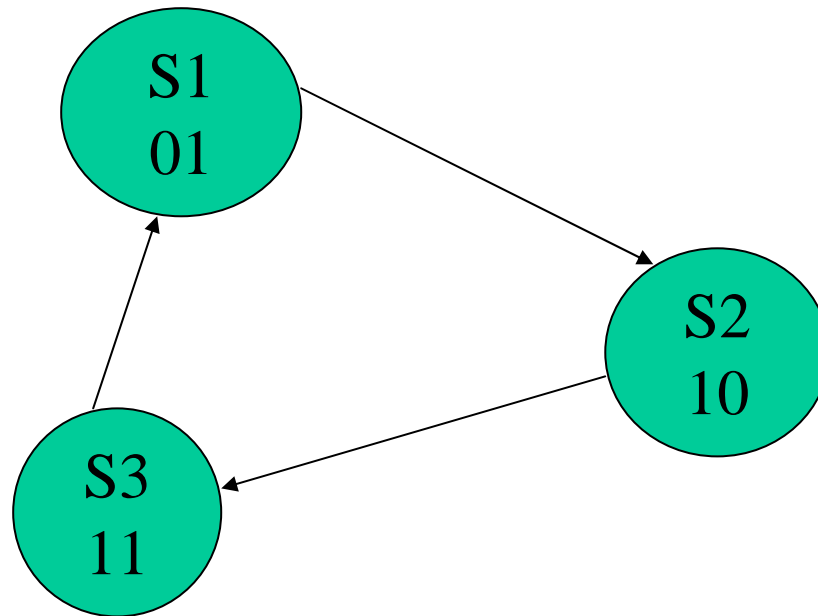
状态机的表达方式

- 1、状态图
- 2、状态表
- 3、流程图

三种表达方法是等价的，可以相互转换

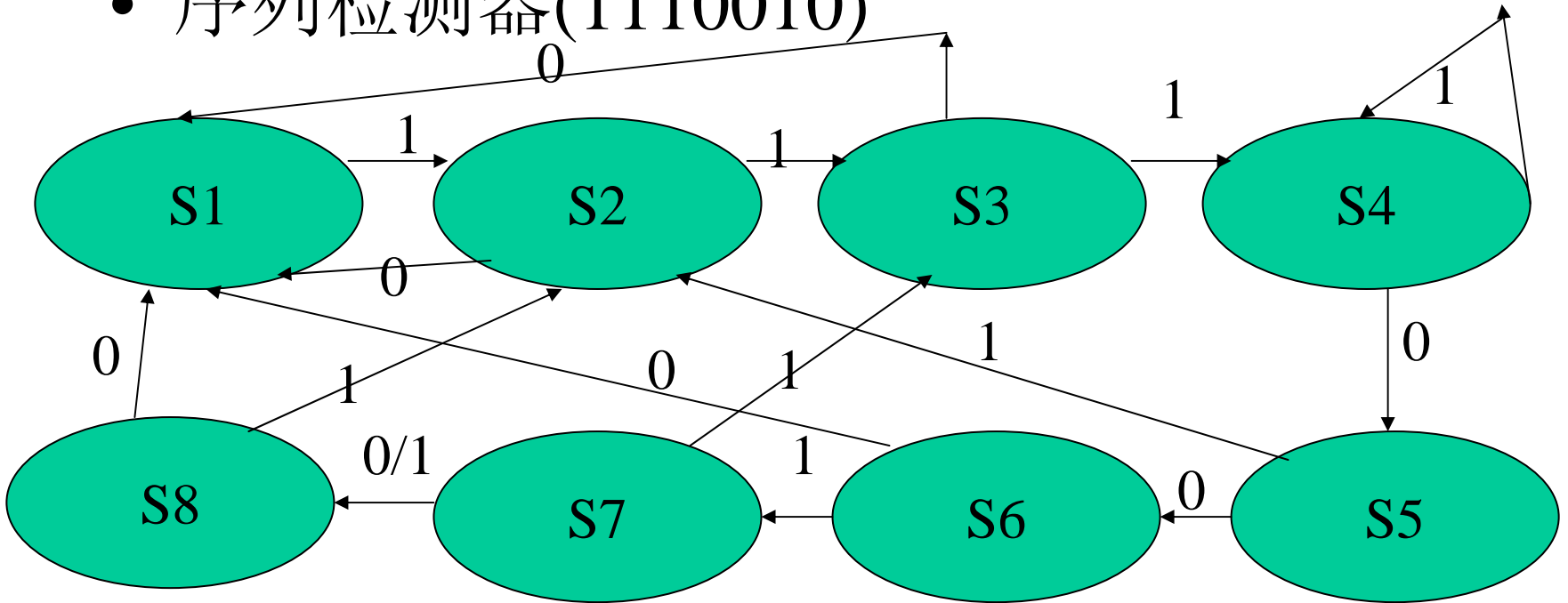
状态机的设计

- 3进制计数器



状态机的设计

- 序列检测器(1110010)



VHDL上机指导

- 编译和仿真工具

OR-CAD或ACTIVE-VHDL

- 本次培训采用ACTIVE-VHDL
- ACTIVE-VHDL自带教程

目录: ..\Active VHDL\book\Avhdl.htm