第3章硬件描述语言VHDL之二

主要内容

- 3.1 概述
- 3.2VHDL基本结构
- 3.3 数据对象类型和运算符
- 3.4顺序语句
- 3.5并行语句
- 3.6子程序
- 3.7程序包与设计库
- 3.8 VHDL描述实例

3.4 顺序语句

- WAIT语句
- 信号赋值语句
- CASE语句
- NEXT语句
- RETURN语句
- 过程调用语句
- REPORT语句

变量赋值语句
IF语句
LOOP语句
EXIT语句
NULL语句
断言语句

顺序语句--等待语句Wait

- 进程在运行中总是处于两种状态之一：执行或挂起。
- 当进程执行到WAIT语句时，就将被挂起来，并设置好再执行的条件。
- WAIT语句可以设置四种不同的条件：无限等待、时间到、条件满足以及敏感信号量变化。这几类条件可以混用。

顺序语句--等待语句Wait

- 语句格式如下：

  （1）WAIT；

  （2）WAIT ON 信号；

  （3）WAIT UNTIL 条件表达式

  （4）WAIT FOR 时间表达式；

顺序语句--断言语句ASSERT

- 断言（ASSERT）语句
- ASSERT　条件　[REPORT 输出信息] [SEVERITY级别]
- 序如P104

顺序语句—信号赋值语句

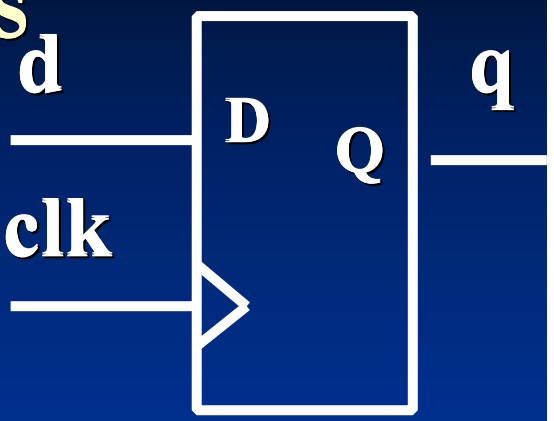
- 信号赋值语句的格式如下：
- 信号量 <= 信号量表达式
- 如：
- **a<= b AFTER 5 ns;**
- 信号赋值语句指定延迟类型，并在后面指定延迟时间。但VHDL综合器忽略延迟特性。

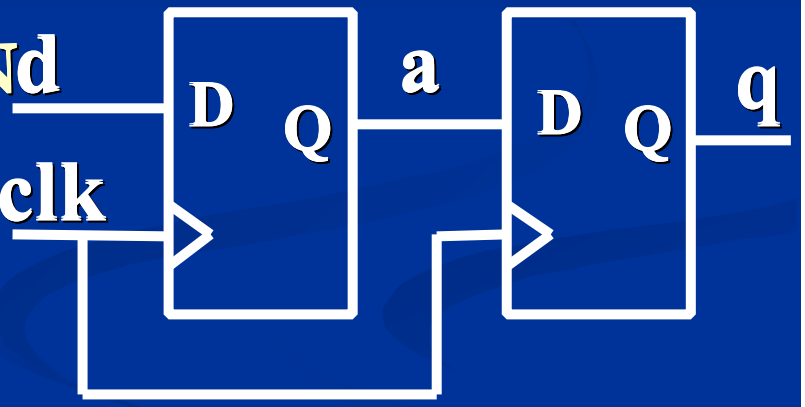
顺序语句—变量赋值语句

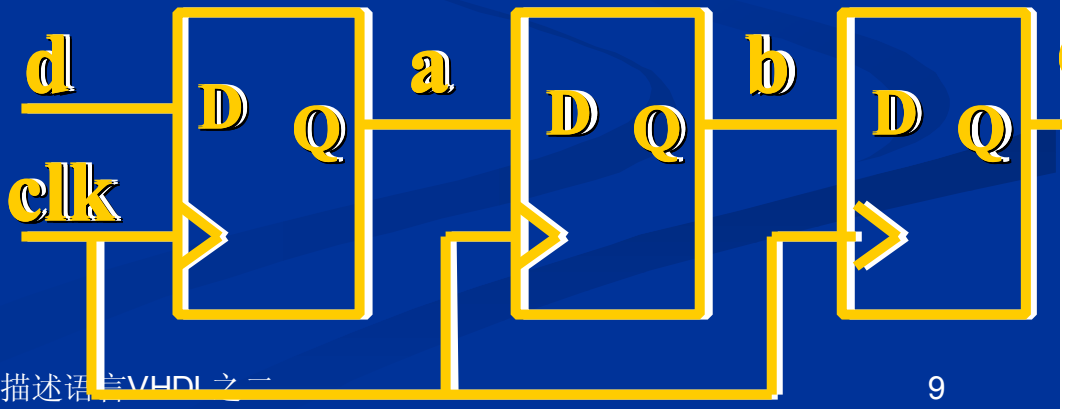
- 变量赋值语句：
- 在VHDL中，变量的说明和赋值限定在进程、函数和过程中。变量赋值符号为“：＝”，同时，符号“：＝”也可用来给变量、信号、常量和文件等对象赋初值。其书写格式为：
- 变量：＝表达式；
- 例如：
- a：＝2；
- d：＝d+e

顺序赋值语句举例

```
ARCHITECTURE reg1 OF reg1 IS
SIGNAL a, b : BIT;
BEGIN
PROCESS (clk)
BEGIN
IF rising_edge(clk) THEN
        a <= d;
        b <= a;
        q <= b;
END IF;
END PROCESS;
END reg1;
```

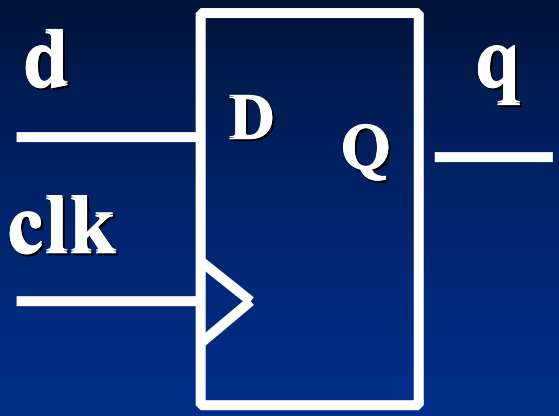


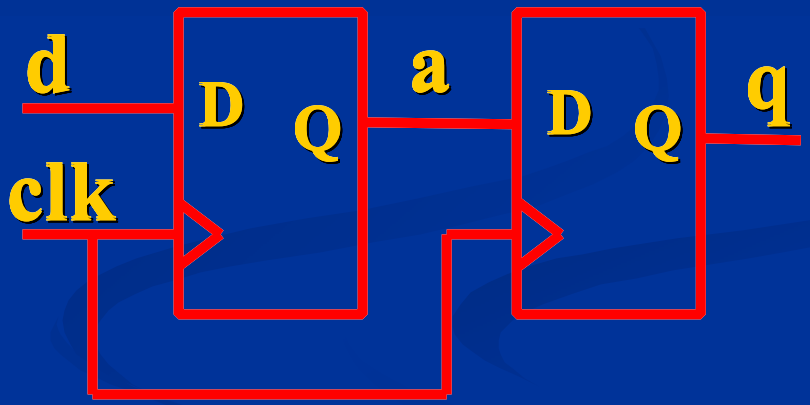


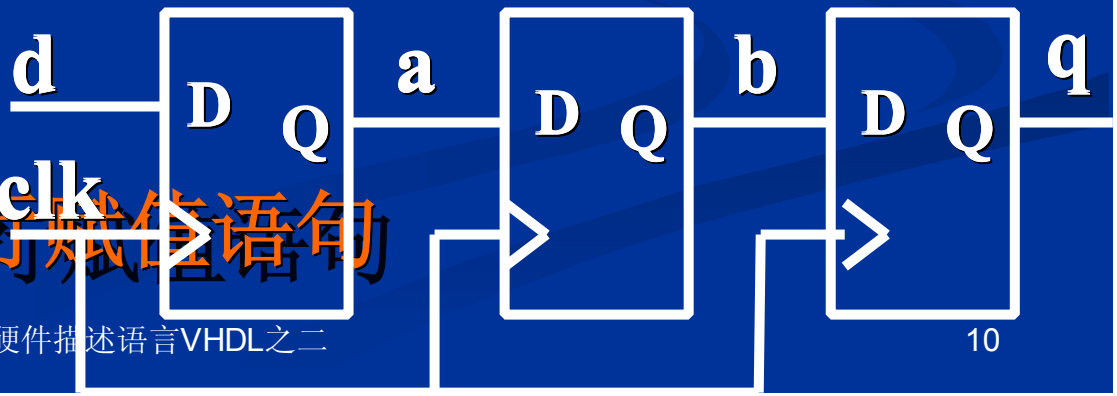


顺序赋值语句举例

```
ENTITY reg1 IS
   PORT (
        d : in BIT;
      clk : in BIT;
        q : out BIT);
  END reg1;
ARCHITECTURE reg1 OF reg1 IS
SIGNAL a, b : BIT;
BEGIN
PROCESS (clk)
BEGIN
IF rising_edge(clk) THEN
a <= d;
b <= a;
END IF;
END PROCESS;
q <= b;
END reg1;
```



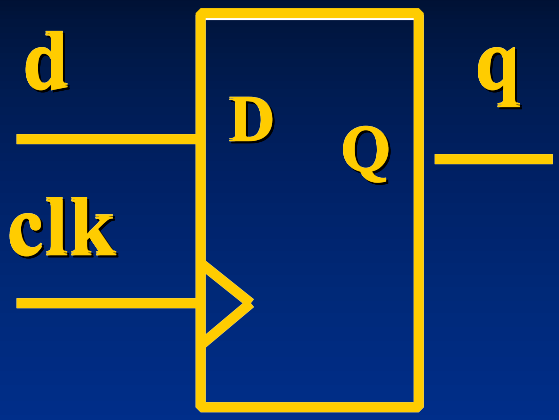




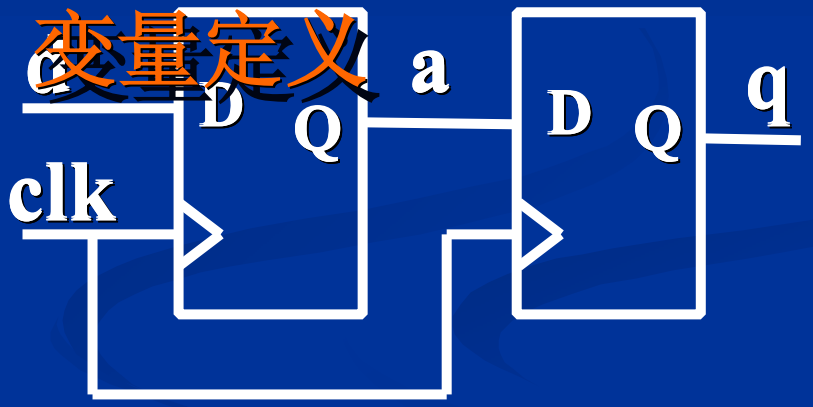
并行赋值语句

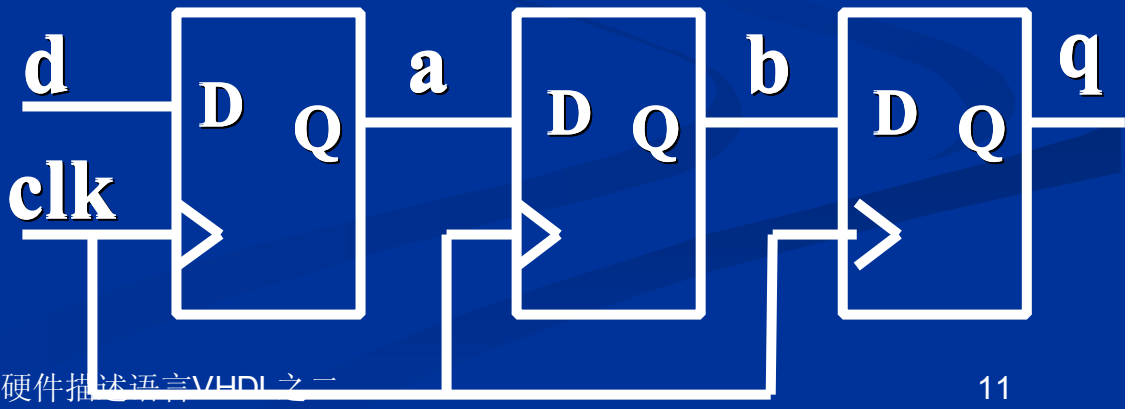
顺序赋值语句举例

```
ENTITY reg1 IS
      PORT ( d : in BIT;
             clk : in BIT;
             q : out BIT);
END reg1;
ARCHITECTURE reg1 OF reg1 IS
VARIABLE a, b : BIT;
BEGIN
PROCESS (clk)
BEGIN
IF rising_edge(clk) THEN
          a <= d;
          b <= a;
          q <= b;
END IF;
END PROCESS;
END reg1;
```



变量定义





IF_THEN_ELSE语句

- 只能在进程内使用
- 至少应有一个条件句，条件句必须由布尔表达式构成。
- 根据条件句产生的判断结果TRUE或FALSE，有条件地选择执行其后的顺序语句。

```
IF 条件句
Then
   顺序语句
ENDIF
```

```
IF 条件句 THEN
   顺序语句
ELSE
   顺序语句
ENDIF
```

```
IF 条件句 Then
   顺序语句
ELSIF 条件句 Then
   顺序语句
ELSE
   顺序语句
ENDIF
```

IF_THEN_ELSE语句

```
mux4_1: PROCESS (a, b, c, d,
    BEGIN
        IF    s = "00" THEN x <= a ;
        ELSIF  s = "01" THEN x <= b ;
        ELSIF  s = "10" THEN x <= c ;
        ELSE   x <= d ;
        END IF;
    END PROCESS mux4_1 ;
```

必须在进程内使用

IF_THEN_ELSE语句

```
ex1: PROCESS (a, b)
        BEGIN
          IF a='1' THEN c<='0';      -- if a and b are
          END IF;                    -- both '1' then
          IF b='1' THEN c<='1';      -- b has priority
          END IF;                    -- so c <= '1';
        END PROCESS ex1;
ex2: PROCESS (a, b)
        BEGIN
          IF b='1' THEN c<='1';      -- if a and b are
          END IF;                    -- both '1' then
          IF a='1' THEN c<='0';      -- a has priority
          END IF;                    -- so c <= '0';
        END PROCESS ex2;
```

CASE语句

- CASE语句的一般格式如下：

  CASE 表达式 IS

  WHEN 表达式值 ＝＞ 顺序语句；
  WHEN OTHERS ＝＞ 顺序语句；

  END CASE；

  当CASE和IS之间的表达式满足指定的值时，程序将执行后面所跟的顺序语句。

CASE语句-实例一

- TYPE enum IS （pick_a, pick_b, pick_c, pick_d）;
- SIGNAL value： enum;
- SIGNAL a, b, c, d, z： BIT;
- CASE value IS
- WHEN pick_a =>
- z<= a;
- WHEN pick_b =>
- z<= b;
- WHEN pick_c =>
- z<= c;
- WHEN pick_d =>
- z<= d;
- END CASE;

CASE语句-实例二

```
mux4_1: PROCESS (a,b,c,d,s)
    BEGIN
        CASE s IS
            WHEN "00"   =>  x <= a;
            WHEN "01"   =>  x <= b;
            WHEN "10"   =>  x <= c;
        WHEN OTHERS =>  x <= d;
        END CASE;
    END PROCESS mux4_1;
```

LOOP语句

- LOOP语句与其他高级语言中的循环语句一样，使程序能进行有规则的循环，循环的次数受迭代算法的控制。一般格式有两种：

LOOP语句-格式（一）

- FOR循环变量：

  [标号]：FOR 循环变量 IN 离散范围
  LOOP

  顺序语句；

  END LOOP [标号]；

例如：

```
VARIABLE a, b : BIT_VECTOR(1 TO 3);
    FOR i IN 1 TO 3 LOOP
     a（i）<= b(i);
    END LOOP;
```

上面循环语句的等价语句如下：

```
    A（1）<= B(1);
    A（2）<= B(2);
    A（3）<= B(3);
```

LOOP语句-格式（一）

- WHILE条件循环

这种LOOP语句的书写格式如下：

[标号]：WHILE 条件 LOOP

顺序语句；

END [标号]；

当条件为真时，则进行循环；当条件为假时，则结束循环。

LOOP语句示例

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
ENTITY shift4 IS
PORT ( shft_lft    : in std_logic;
       d_in   : in std_logic_vector(3 downto 0);
       q_out : out std_logic_vector(7 downto 0));
END shift4;
ARCHITECTURE logic OF shift4 IS
BEGIN
PROCESS(d_in, shft_lft)
VARIABLE shft_var : std_logic_vector(7
DOWNTO 0);
```

```
BEGIN
   shft_var(7 downto 4) := "0000";
   shft_var(3 downto 0) := d_in;
IF shft_lft = '1' THEN
   FOR i IN 7 DOWNTO 4 LOOP
        shft_var(i) := shft_var(i-4);
   END LOOP;
   shft_var(3 downto 0) := “0000”;
ELSE          shft_var := shft_var;
END IF;
   q_out <= shft_var;
END PROCESS;
END logic;
```

NEXT--实例

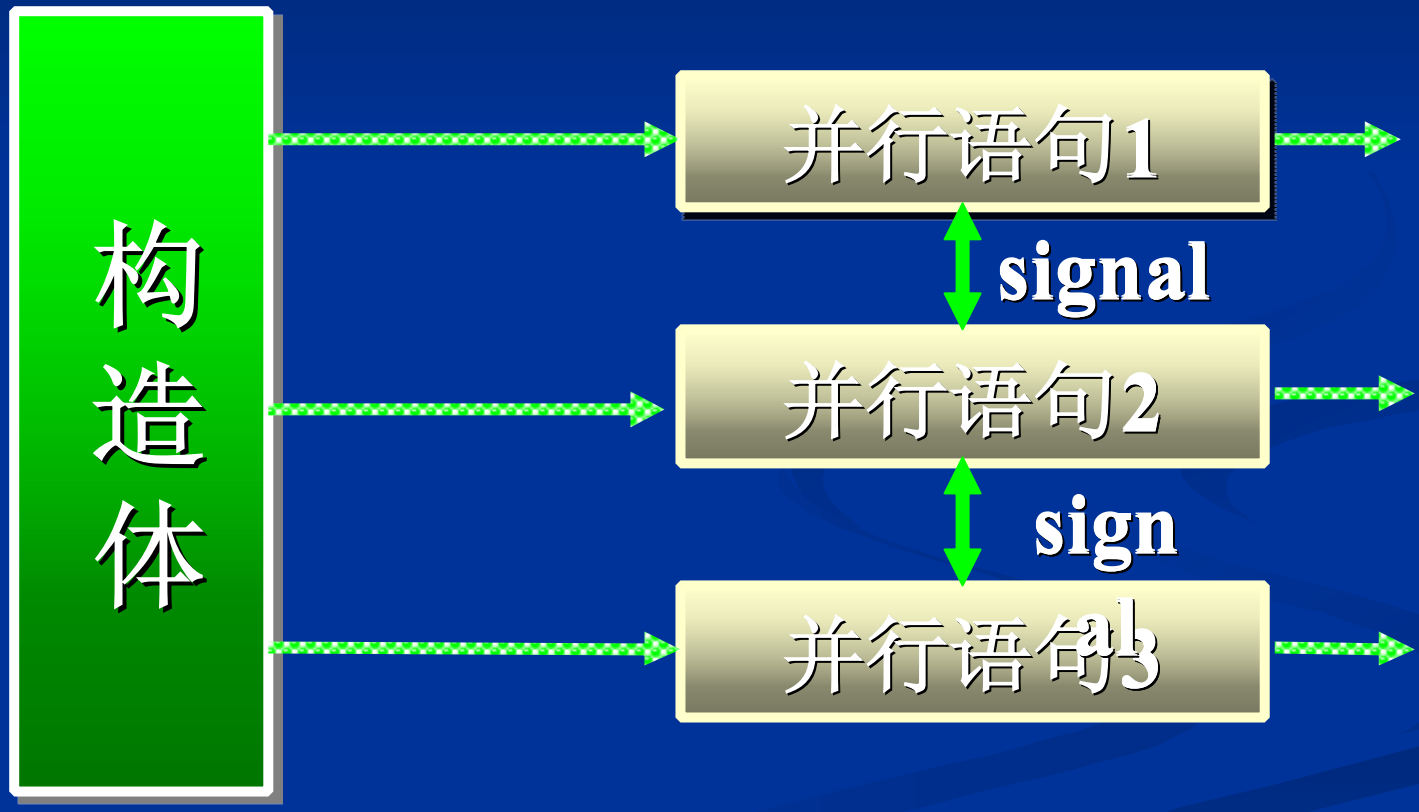
- SIGNAL a，b，copy_enable：BIT_VECTOR(1 TO 8);
- …
- a<= “ 00000000”;
- …
- ——b被赋了一个值，如“11010011”
- …
- FOR i IN 1 TO 8 LOOP
- NEXT WHEN copy_enableE(i) ='0';
- a（i）<= b(i);
- END LOOP;

NULL语句

- NULL语句表示没有动作发生。NULL语句一般用在CASE语句中以便能够覆盖所有可能的条件。

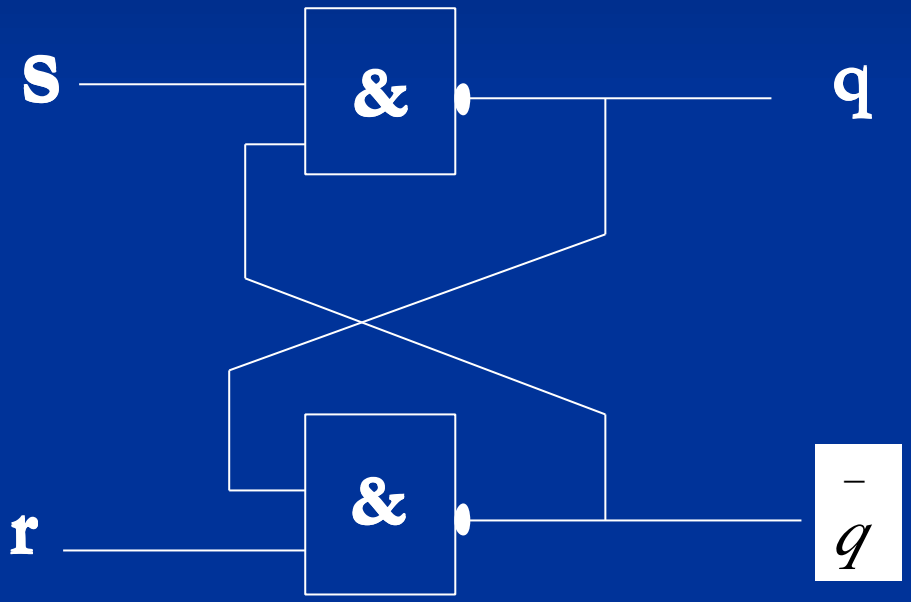
返回

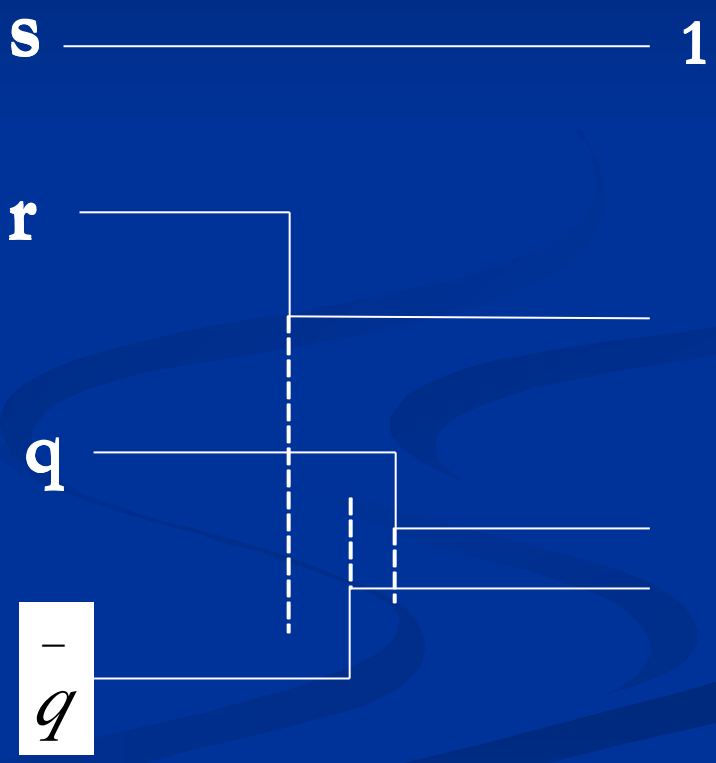
3.5并行语句



并行语句-实例

- 基本的RS触发器：





设仿真时，r, s均为1，nq初始值为0，某一时刻，r由1变为0。

```
ENTITY rs_ff  IS
    PORT (r, s: IN Bit;   q,nq: BUFFER Bit)
  END rs_ff;
ARCHITECTURE funct_rsff  OF rs_ff
  BEGIN
     q<=s NAND nq;
     nq<=r NAND q;
  END funct_rsff;
```

并行语句

- 主要的并行语句有以下几种：
- 信号赋值语句
- 块（BLOCK）语句
- 进程（PROCESS）语句
- 生成（GENERATE）语句
- 器件（COMPONENT）和器件例化（COMPONENT_INSTANT）语句

并行信号赋值-条件型

- 条件型赋值语句：

  信号名<=表达式1 WHEN（条件1）ELSE

  .

  .

  表达式N-1 WHEN（条件N-1）ELSE

  表达式N；

（根据条件满足与否决定哪个表达式的值赋给待赋值的信号）　程序参见101

并行信号赋值-选择型

选择信号赋值语句的格式为：

WITH 选择表达式 SELECT

信号名<=表达式1 WHEN（条件1）选择值1

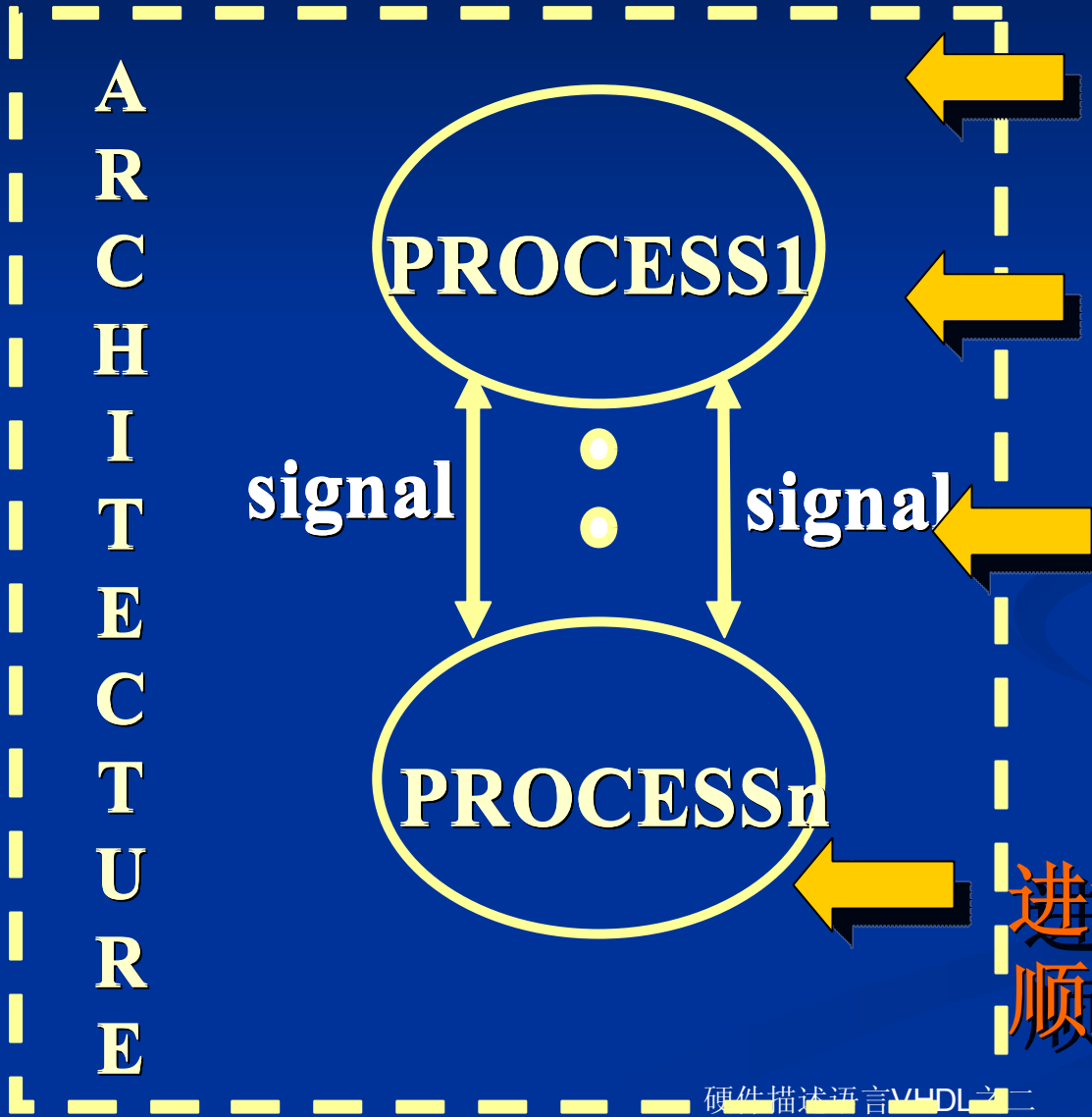
.

.

表达式N WHEN 选择值2

程序示例如P102

并行语句-进程语句



一个构造体可以有多个进程语句

进程和进程之间是并行的

进程和进程之间的数据交换通过信号完成

进程内部是顺序语句

进程语句举例

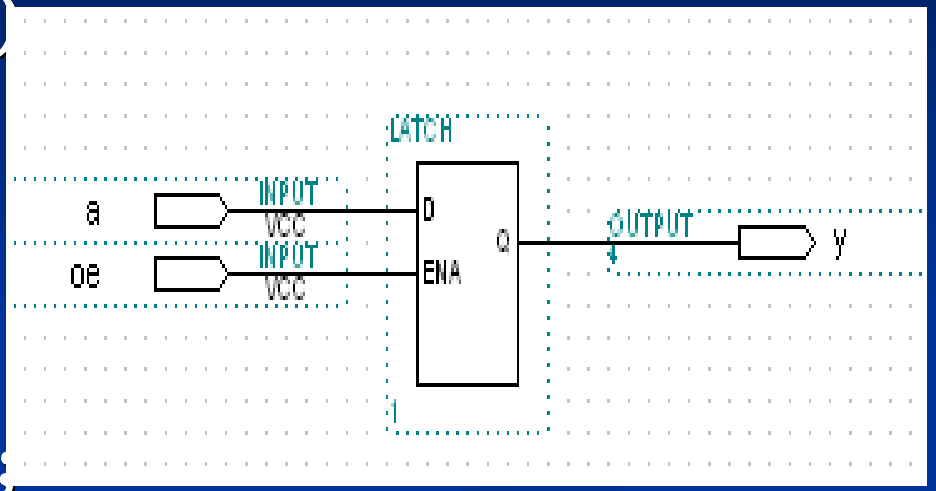
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY if_case IS PORT
     ( a, b, c, d : IN Std_Logic;
      sel : IN Std_Logic_Vector(1
    downto 0);
      y, z : OUT Std_Logic);
END if_case;
ARCHITECTURE logic OF if_case
    IS
BEGIN
if_label: PROCESS(a, b, c, d, sel)
BEGIN
IF sel="00" THEN  y <=
ELSIF sel="01" THEN  y
ELSIF sel="10" THEN  y
ELSE  y <= d;
END IF;
END PROCESS if_label;
```

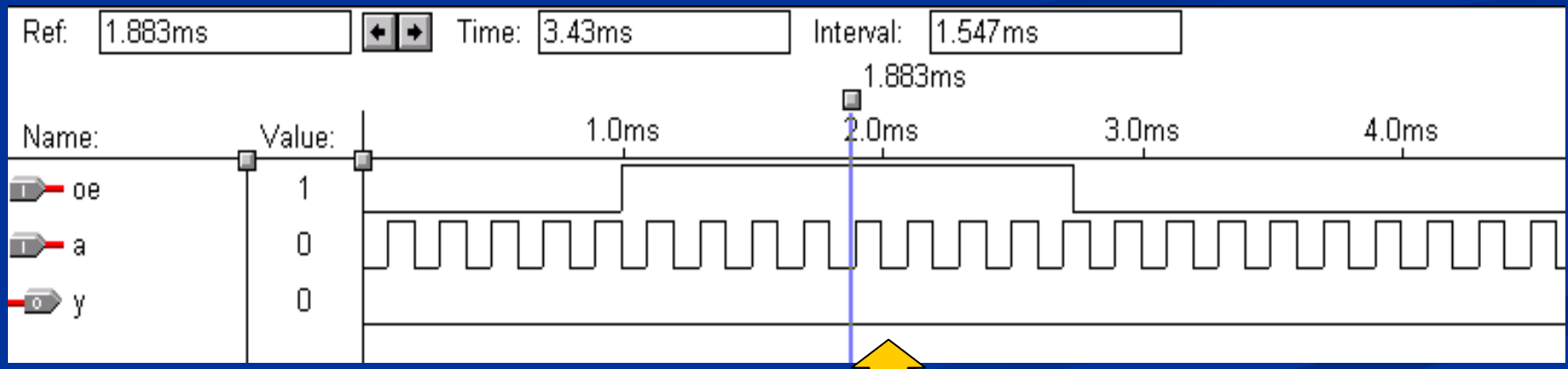
```
case_label:
PROCESS(a, b, c, d, sel)
BEGIN
CASE sel IS
WHEN "00" =>z <= a;
WHEN "01" =>z <= b;
WHEN "10" =>z <= c;
N "11" =>z <= d;
N OTHERS =>z
;
CASE;
D PROCESS
se_label;
END logic;
```

进程的运行依赖于敏感表内参数的变化并发的

敏感表举例

```
latchlabel: PROCESS(oe)
BEGIN
    IF oe='1' THEN
        y <= a;
    END IF;
END PROCESS if_label;
```

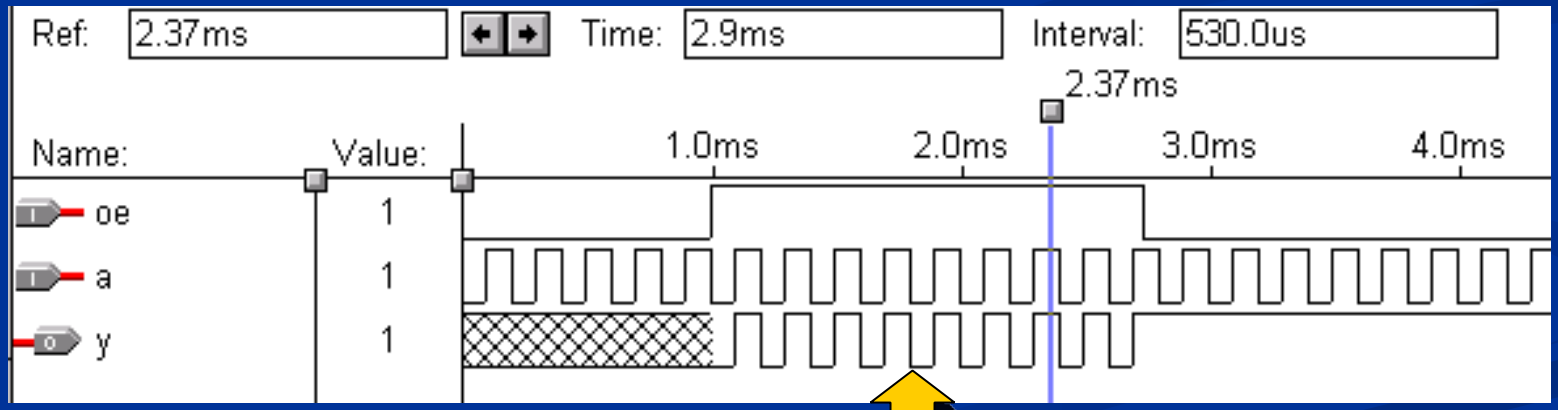




仿真结果错误

敏感表举例

```
latchlabel: PROCESS(oe,a)
BEGIN
   IF oe='1' THEN
        y <= a;
   END IF;
END PROCESS if_label;
```



仿真结果正确

并行语句-元件例化（COMPONENT_INSTANT）语句

- COMPONENT语句的基本格式如下：

```
COMPONENT 元件名
    GENERIC说明；    --参数说明
    PORT说明；       --端口说明
END COMPONENT;
```

- COMPONENT and2
  PORT（a，b：IN BIT；
  c： OUT BIT）；
  END COMPONENT；

...

SIGNAL x，y，z： BIT；

...

u1：and2 PORT MAP(x，y，z)； ——位置映射

u2：and2 PORT MAP(a=>x，c =>z，b =>y)； ——名称映射

u3：and2 PORT MAP(x，y，c =>z) ——混合形式

并行语句-生成（Generate）语句

- 标号：FOR 变量 IN 不连续区间
  GENERATE
  并行处理语句
  END GENERATE [标号];

- 标号：IF 条件 GENERATE
  并行处理语句
  END GENERATE [标号];

- 例如：

```
SIGNAL a，b：BIT_VECTOR（3 DOWN TO 0）；
SIGNAL c ：BIT_VECTOR（7 DOWN TO 0）；
SIGNAL x ：BIT；
    …
GEN_LABEL：FOR i IN 3 DOWNTO 0
GENERATE
    c( 2 * i + 1) < = a(i) NOR x;
    c( 2 * i ) < = b(i) NOR x;
END GENERATE GEN_LABEL;
```

并行语句--块（BLOCK）语句

- 块（BLOCK）语句：
- 块语句把若干条并行语句包装在一起，表示一个子模块
- 结构体本身就等价于一个子模块。
- 可以将一个较为复杂的结构体划分成几个块，分别进行描述
- 所以，BOLCK语句提供了在一个实体（结构体）中进行层次化设计的方法。块语句的格式如下：

- 标号：**BLOCK**

  [类属说明与映射]

  [端口说明与映射]

  [说明部分]

  BEGIN

  [并行语句]

  END BLOCK 标号;

块头主要用于信号的映射及参数的定义，通常通过**GENETIC**语句、**GENETIC_MAP**语句、**PORT**和**PORT_MAP**语句来实现。

- 半加器的结构体可以改写为：

```
ARCHITECTURE describ_ha OF half_adder is
BEGIN
    half_blk: BLOCK
     PORT(x,y: IN Bit;
              z,f: OUT Bit);--定义块的局部端口
     PORT MAP( a,b ,s,c);--说明块端口与外部信号的连接关系
     BEGIN
      z<=x XOR y;
```

```
    f<= x  AND y;
  END BLOCK half_blk;
   END describ_ha;
```

通过BLOCK将两条并行语句封装成一个模块。

返回

3.6子程序—函数定义与引用

函数的作用是求值，它有若干参数输入，但只有一个返回值作为输出。

定义一个函数的一般格式：

FUNCTION 函数（参数表） **RETURN** 数据类型 **IS**

[说明语句；]

BEGIN

顺序语句；

END[FUNCTION] [函数名];

参数表中需说明参数名及其数据类型

表示返回值的类型，、参数类别（信号或常量也称为函数的类型

顺序语句为函数体，定义函数的功能

- 定义一个函数max,从两个整数中求最大值
  程序参见P108

- 函数通常在一个顺序语句的表达式中被引用。如对D触发器功能的下列描述：

子程序—过程定义与引用

- 过程通过参数进行内外信息的传递。
- 参数需说明类别（信号、变量或常量）、类型及传递方向（IN、OUT或INOUT，默认方向为IN）
- 过程定义的一般格式为

- **PROCEDURE** 过程名（参数表） **IS**

  [说明语句；]

  BEGIN

  顺序语句；

  END[RORCEDURE][过程名]；

  过程的引用只需要直接写过程名及相应的参数　实例参见**P109**

子程序-子程序重载

- 指两个或多个子程序使用相同的名字，当多个子程序重名时，下列因素将决定某次调用使用的是哪一个子程序：
- 子程序调用中出现的参数数目
- 调用中出现的参数类型
- 调用中参数采用名字关联方式时形参的名字
- 子程序为函数时返回值的类型　举例如110

子程序-子程序重载

- VHDL中，逻辑运算符、关系运算符和算术运算符都是函数，且可重载
- 运算符的重载与重载函数遵循同样的规则。
- 同运算符对应的函数名为：

  “运算符符号”

如需要定义一种三态类型bit3则：

TYPE bit3 IS(‘0’,‘1’，‘Z’）

并要给出这种类型对象的“与”运算规则，则可以不改变“与”运算符（AND）而通过重载而实现，如P111

3.7 程序包（PACKAGE）

- 程序包由标题和包体两部分组成，其结构如下：

标题部分

```
PACKAGE 程序包名 IS
    --- 说明语句
END 程序包名
```

包体部分

```
PACKAGE BODY 程序包名 IS
    --- 说明语句
END BODY;
```

常用的程序包如下：

- （1）STANDARD程序包：

  LIBRARY WORK.STD;

  USE STD.STANDARD.ALL;

（2）STD_LOGIC_1164程序包

（3）STD_LOGIC_UNSIGNED程序包

（4）STD_LOGIC_SIGNED程序包

常用的库如下：

- （1）STD库
- （2）WORK库
- （3）IEEE库

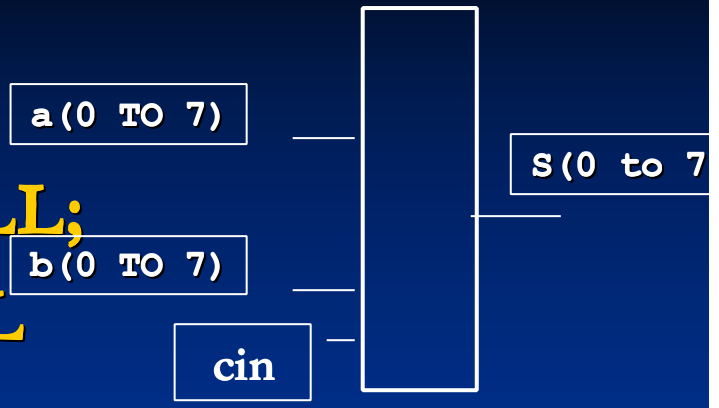
返回

3.8 VHDL描述实例

- 组合逻辑电路描述
- 时序逻辑电路描述
- 状态机的描述

组合电路举例

多位加法器




```
LIBRARY IEEE;
USE IEEE.Std_logic_1164.ALL;
USE IEEE.Numeric_Std.ALL
ENTITY adder IS
PORT (a, b: IN std_logic_vector(0 TO7);
          cin:  IN std_logic;
          s: OUT std_logic_vector(0 TO7));
END adder;
ARCHITECTURE  behav_adder OF adder IS
BEGIN
        s<=('0'&a)+('0'&b)+("00000000"&cin);
END behav_adder;
```

组合电路举例 多位比较器

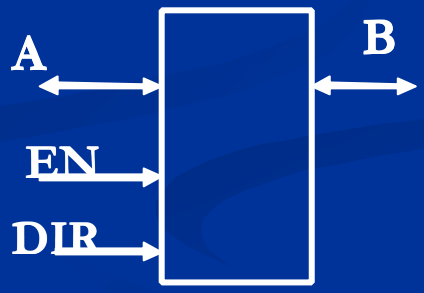
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY compare IS
PORT (
        a, b: IN std_logic_vector(0 TO 7);
        a_eq_b, a_greater_b, a_less_b, OUT std_logic);
END compare;
ARCHITECTURE archcompare OF compare IS
BEGIN
    a_eq_b <='1'        when a=b ELSE '0';
    a_greater_b <='1'   when a>b ELSE '0' ;
    a_less_b <='1'      when a<b ELSE '0';

END archcompare;
```

组合电路举例三态与门

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ts_andgate IS
PORT (en,a, b: IN std_logic;
        f: OUT std_logic);
END ts_andgate;
ARCHITECTURE behav_tsgate OF ts_andgate IS
BEGIN
    PROCESS( en,a,b）
   BEGIN
    IF  en='1' THEN
        f<=a  AND  b;
     ELSE
        f<='Z';
     END  IF
    END   PROCESS
END behav_tsgate;
```

组合电路举例 双向总线驱动器

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY bidir IS
PORT (a, b: INOUT std_logic_vector(0 TO 7);
        en,dir: OUT std_logic);
END bidir;
ARCHITECTURE behav_bidir OF bidir IS
BEGIN
   PROCESS( en,dir,a）
  BEGIN
   IF  (en='0') THEN
       b<="ZZZZZZZZ";
   ELSE(en='1' AND dir='1')  THEN
       b<=a;
   END  IF
   END   PROCESS
```



```
PROCESS( en,dir,a）
   BEGIN
    IF  (en=‘0’) THEN
        a<=“ZZZZZZZZ”;
    ELSE(en=‘1’ AND dir=‘1’)  THEN
        a<=b;
    END  IF
   END   PROCESS
 END behav_bidir
```

返回

时序电路举例

D锁存器

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY d_latch IS
PORT (clk, d : in std_logic;
          q ,nq: out std_logic);
END d_latch;
ARCHITECTURE behav _dlatch OF d_latch IS
BEGIN
PROCESS (clk,d)
BEGIN
    IF clk = '1' THEN
   q <= d;
   nq <=NOT d;
END IF;
END PROCESS;
END behav _dlatch ;
```

时序电路举例 主从JK触发器

主从JK触发器在时钟脉冲（CLK）的上升沿被触发，但输出却要延迟到时钟脉冲的下降沿才会发生变化

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY msjk_ff IS
PORT (clk, j,k : IN std_logic;
        q ,nq: BUFFER std_logic);
END msjk_ff;
```

```
ARCHITECTURE behav _msjkff OF msjk_ff IS
BEGIN
PROCESS (clk)
    VARIABLE mid_q: Std_Logic;
BEGIN
    IF clk = '1' THEN
   mid_q:=(j AND nq) OR  (NOT k AND q);
   ELSE
  q<=mid_q;
   nq <=NOT mid_q;
END IF;
END PROCESS;
END behav _msjkff ;
```

时序电路举例

双向移位寄存器

该双向移位寄存器除具有8位二进制数据存储功能外，还具有左移、右移、并行置数和同步复位的功能

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY srg IS
PORT (reset,clk:  IN Std_logic;
        data: IN Std_logic_Vector(7 DOWNTO 0);
        sl_in,sr_in: IN Std_logic;
        mode: IN Std_logic_Vector(1 DOWNTO 0);
        q: BUFFER Std_logic_Vector(7 DOWNTO 0);
END srg;
```

时序电路举例 双向移位寄存器

```
    ARCHITECTURE behav _srg OF srg IS
BEGIN
PROCESS
BEGIN
    WAIT ON clk UNTIL Rising_edge(clk);
 IF reset = '1' THEN
   q<="00000000";   ----同步复位
  ELSE
   CASE mode IS;
     WHEN "01"=>  q<=sr_in &q(7 DOWNTO 1);--右移
     WHEN "10" =>  q<=q(6 DOWNTO 0) &sl_in; --左移
     WHEN "11" =>   q<=data;          --并行输入（同步置位）
     WHEN OTHERS => NULL        --空操作，即保持
    END CASE;
 END IF;
END PROCESS;
END behav_srg;
```

时序电路举例 可逆计数器

该计数器为异步复位（RESET 端，高电平有效）、异步预置（LD端，高电平有效）的模10可逆计数器。
计数方式控制端UP/DOWN=1进行加法计数，UP/DOWN=0进行减法计数

```
LIBRARY IEEE;
USE IEEE.Std_logic_1164.all;
USE IEEE.Numeric_Std.all;
ENTITY counter IS
PORT (reset, ld:  IN Std_logic;
        clk, up_down: IN Std_logic);
        data: IN Std_logic_Vector(3 DOWNTO 0);
        q: BUFFER Std_logic_Vector(3 DOWNTO 0);
        bo,co: OUT Std_logic);
END counter;
```

时序电路举例 可逆计数器

```
ARCHITECTURE behav _counter OF counter IS
BEGIN
 co<='1' WHEN (q="1001"  AND up_down='1')
  ELSE '0';
 bo<='1'  WHEN(q="0000"  AND up_down='0')
  ELSE'0';
PROCESS（clk, reset, ld)
BEGIN
   IF reset='1' THEN
       q<='0000';
  --异步复位
   ELSEIF  ld='1'  THEN
     q<=data;
  --异步预置
   ELSEIF(clk'Event  AND clk='1') THEN
```

```
IF  up_down='1'  THEN                          -- 加计数
      IF  q="1001" THEN
          q<="0000";
      ELSE
          q<=q+1;
      END IF;
ELSE                                           --减计数
      IF q="0000" THEN
         q<='1001";
     ELSE
        q<=q-1;
    END IF;
  END IF;
END PROCESS;
END behav _counter ;
```

返回

状态机的描述

- 状态机的描述通常分成状态转换与电路输出两部分。

PS \ X	0	1
S0	S1/0	S3/0
S1	S2/0	S0/0
S2	S3/0	S1/0
S3	S0/0	S2/0