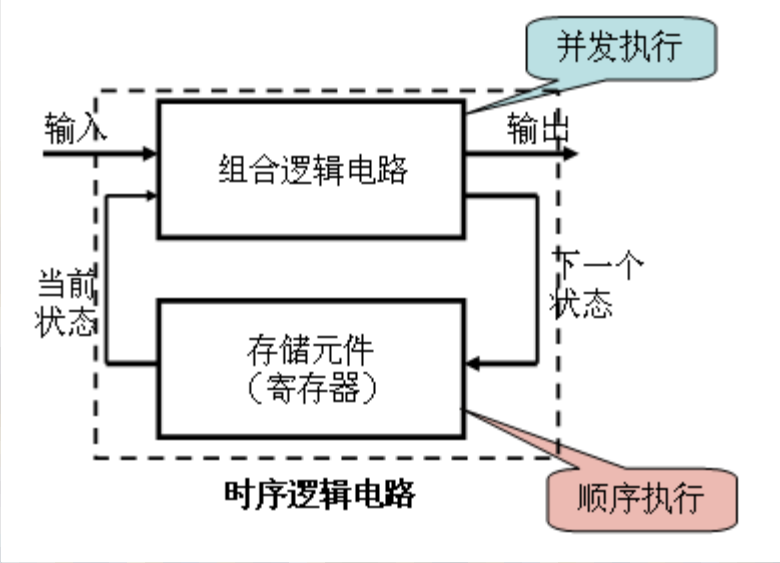
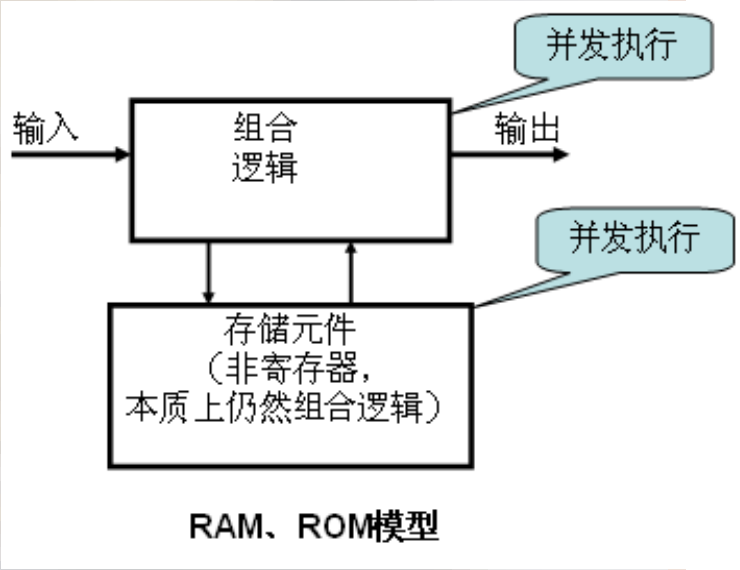
第5章 并发代码（Concurrent）

- 现有的软件编程语言，如机器语言、汇编语言、结构化语言如C语言、面向对象语言乃至形式化语言—本质上都是依据冯。诺依曼模型，由CPU以指令方式串行执行—语句都是逐行顺序执行的，称之为程序。
- VHDL代码--模拟硬件电路的实际执行方式，所有的逻辑门在任何时刻都处于执行状态，称之为代码--按执行顺序可分为两大类：
  - 并发（Concurrent）代码（硬件电路本质）
  - 顺序（Sequential）代码

5.1 并发执行和顺序执行

- 组合逻辑　VS　并发执行代码
- 时序逻辑　VS　顺序执行代码

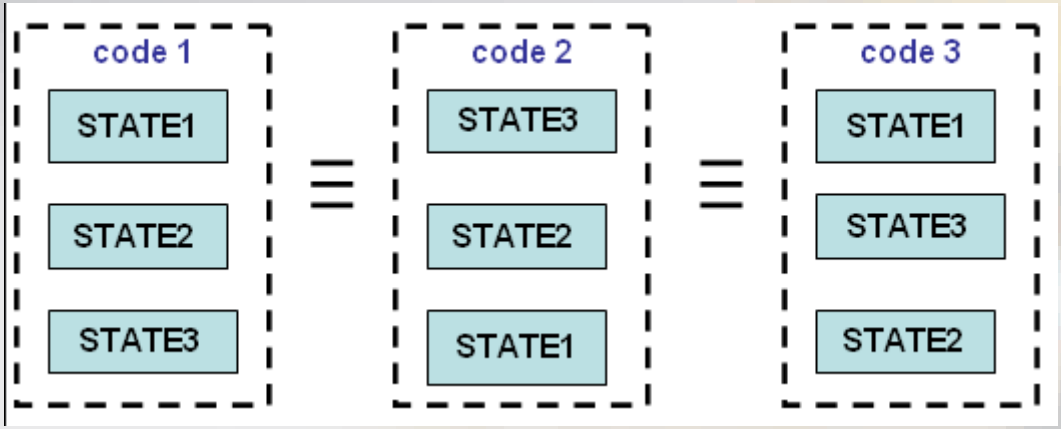




VHDL代码本质上是并发执行的。但在另一方面，为了实现某些本质上具有串行执行特点的功能，如时序相关的功能，VHDL代码也提供了能够顺序执行的语句，如： process/function/procedure语句或结构。

- process/function/procedure内部的代码才是顺序执行的（往往在设计时序电路时），但这些语句或结构之间仍然是并发执行的。
- 并发代码又称“数据流代码”。

例：一段包含3个并发描述语句的代码



并发代码要点：

1、除卫氏块（guarded block）外，仅使用并发描述语句无法实现同步时序电路（异步时序电路是可以的）；

2、通常只使用并发描述语句实现组合逻辑电路；

顺序代码要点：

1、要实现时序逻辑电路，必须使用顺序描述语句；

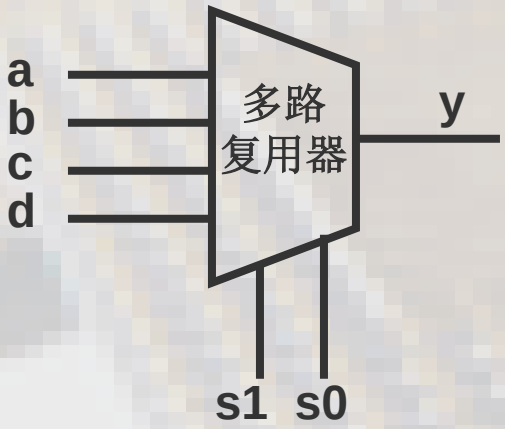
2、顺序代码可同时实现组合逻辑电路与时序逻辑电路。

常用的并发描述语句，通常位于process、function和procedure之外，包括：

- 运算操作符：逻辑运算、算术运算等
- WHEN语句（when/else或with/select/when）
- 块（block）语句、
- 生成（GENERATE）语句。

5.2 使用运算操作符—建立并发代码的最基本的方法

- 运算操作符：见表5.1
- 运算操作符可以实现任何组合逻辑电路
- 例：多路复用器



•功能描述：根据选择位s1和s0的值，从四路输入中选择其中一路输出。

•真值表或表达式分析

•真值表

输入	选择位 s1	选择位 s0	输出
a/b/c/d 四路并行输入	0	0	a
	0	1	b
	1	0	c
	1	1	d

- 实现代码：

```
1----------------------------------------------------
2     library ieee;
3     use ieee.std_logic_1164.all;
4----------------------------------------------------
5     entity mux is
6          port(  a,b,c,d,s1,s0: IN std_logic;
7                      y: OUT std_logic );
8     end mux;
9-----------------------------------------------
10 architecture pure_logic of mux is
11 begin
12    y<=(a AND  NOT s1 AND NOT s0) OR
13          (b AND NOT  s1 AND s0) OR
14          (c AND s1 AND NOT s0) OR
15          (d AND s1 AND s0);
16  end pure_logic;
17 ------------------------------------------------
```

波形图

5.3 WHEN语句

- 一种基本的并发代码描述语句
- 两种形式：
  - WHEN/ELSE（又称simple WHEN）
  - WITH/SELECT/WHEN（又称selected WHEN）
- WHEN/ELSE语句的语法结构：

```
assign WHEN condition ELSE
assign WHEN condition ELSE
....;
```

- WITH/SELECT/WHEN语句的语法结构:

```
WITH identifier SELECT
        assignment WHEN value,
        assignment WHEN value,
        .....;
```

注意：必须考虑所有可能出现的条件(condition)，需经常使用关键字OTHERS。若某些条件下不需要进行任何操作，需使用UNAFFECTED。

- **WHEN/ELSE用法例子：**

表达式

```
outp<="000" when (inp='0' OR reset='1')  else
        "001" when ct1='1' else
        "010";
```

- **WITH/SELECT/WHEN用法例子：**

control值

标点

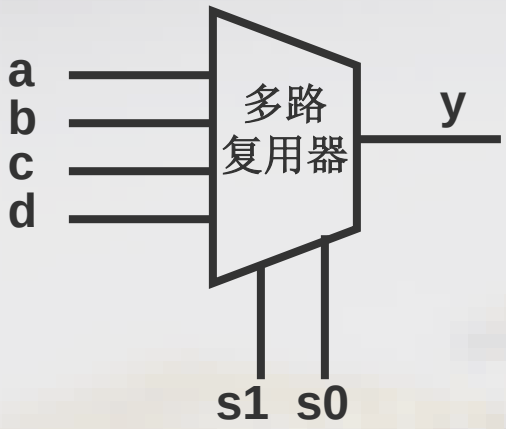
```
with control SELECT
  outp<="000" when reset,
        "111"  when set,
        unaffected when others;
```

control取其他值时
output值保持不变

- **WHEN value的三种描述方式：**

```
when value                    ---对单个值进行判断
when value1 to value2         ---对取值范围进行判断，
                              ---多用于枚举类型；
when value1 | value2 | ...    ---对多个值进行判断
```

多路复用器的另外两种实现方式



- **功能描述：根据选择位sel(1:0)的值，从四路输入中选择其中一路输出。**
- **真值表（略）**
- **两种实现方案：**

  when/else语句（simple WHEN）

  with/select/when语句（selected WHEN）

方案1： when/else方式：

```
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------
entity mux is
   port( a,b,c,d: IN std_logic;
         sel: IN std_logic_vector(1 downto 0);
         y: OUT std_logic );
    end mux;
-----------------------------------------
architecture mux1 of mux is
 begin
    y<= a when sel="00" else
         b when sel="01" else
         c  when sel="10" else
         d ;
 end mux1;
-----------------------------------
```

方案2： with/select/when方式：

```
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------
entity mux is
   port( a,b,c,d: IN std_logic;
         sel: IN std_logic_vector(1 downto 0);
         y: OUT std_logic );
    end mux;
-----------------------------------------
architecture mux2 of mux is
 begin
     with sel  select
         y<=a when "00",
              b when "01",
              c  when "10",
              d  when OTHERS;
 end mux2;
-----------------------------------
```

不能是d when "11";

注意：std_logic的取值还可能是“zz”等，因此必须others 或 else！

将sel信号声明为INTEGER类型时，实现方式如下：

```
-----------------------------------
library ieee;
 use ieee.std_logic_1164.all;
-----------------------------------
entity mux is
    port( a,b,c,d: IN std_logic;
        sel: IN INTEGER range 0 to 3;
        y: OUT std_logic );
      end mux;
```

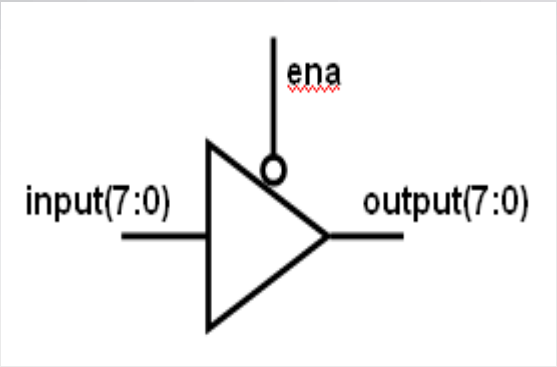
----方案1： when/else方式：--------

```
architecture mux1 of mux is
 begin
   y<= a when sel=0 else
        b when sel=1 else
        c  when sel=2 else
        d ;
 end mux1;
-------------------------
```

---方案2： with/select/when方式：

```
architecture mux2 of mux is
 begin
     with sel  select
         y<=a when 0,
              b when 1,
              c  when 2,
              d  when 3; ---3与OTHERS等效;
 end mux2;
-----------------------------------
```

例：三态缓冲器



•**功能描述：ena为低电平时，输出等于输入，否则输出为“*ZZZZ_ZZZZ*”（高阻态）。**

```
library ieee;
use ieee.std_logic_1164.all;
---------------------------------------
entity tri_state is
   port( ena: IN std_logic;
         input: IN std_logic_vector(7 downto 0);
         output: OUT std_logic_vector(7 downto 0) );
    end tri_state;
---------------------------------------
architecture tri_state of tri_state is
begin
    output<= input when ena='0' else
                  (OTHERS=>'Z') ;
end tri_state;
```

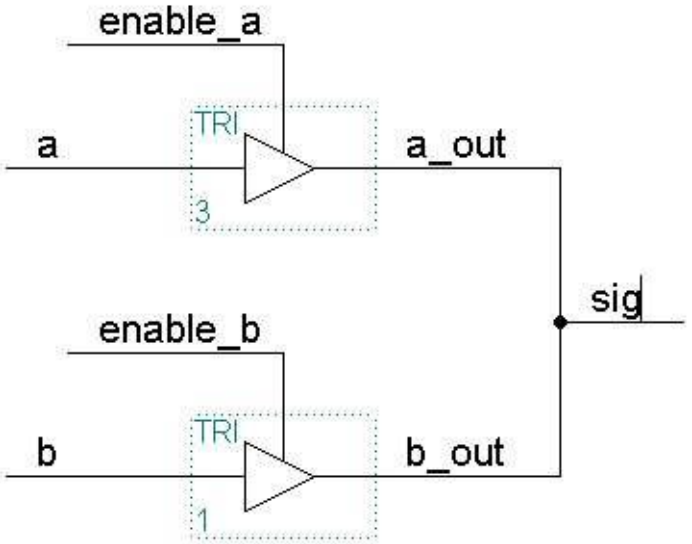
补充知识点：三态缓冲器总线结构与多驱动信号

定义：给一个信号赋值，即为该信号创建一个驱动器（驱动信号）。多个进程或并发语句给同一个信号赋值，则该信号为多信号源驱动。

例：

```
a_out <= a  when enable_a else  'Z' ;
b_out <= b  when enable_b else  'Z' ;
process ( a_out)
begin
    sig <= a_out ;
end   process ;

process ( b_out )
begin
    sig <= b_out ;
end  process ;
```



5.4 生成语句(GENERATE)—可用于并发描述

生成语句的作用：复制建立某项操作的 0 个或多个备份，这些备份并行地执行某项操作。（并行结构，与先后顺序无关；而顺序描述语句中循环执行某项操作的LOOP语句则必须顺序的执行这些操作。）

分为两类：

for --- generate：采用一个离散的范围决定备份的数目。

If --- generate：有条件地生成 0 个或 1 个备份。

1、for --- generate 语句

语法：

标号：for 循环变量 in **range** generate

{ 并行语句 }

end generate [标号];

range：整数表达式 to 整数表达式

整数表达式 downto 整数表达式

for --- loop 语句与 for --- generate 的比较：

例：用生成语句创建多个备份

```
component   comp
      port (x : in  bit ;
            y : out  bit) ;
end  component ;
          ----
signal  a ,  b  : bit_vector (0  to  7) ;
          ----
gen : for  I  in  a'range  generate
   u :    comp   port  map (x => a( I ), y => b( I )) ;
end  generate  gen ;
```

例：4位移位寄存器

```
library ieee;
use ieee.std_logic_1164.all;
entity shift is
   port(a,clk:in std_logic;
        b:out std_logic);
end shift;
architecture gen_shift of shift is
   component dff
      port(d,clk:in std_logic;
           g:out std_logic);
   end component;
   signal z:std_logic_vector(0 to 4);
begin
   a(0)<=a;  b<=z(4);
   g1:for i in 0 to 3 generate
      dffx:dff port map(z(i), clk, z(i+1));
   end generate;
end gen_shift;
```

4位移位寄存器的等效描述：

```
library ieee;
use ieee.std_logic_1164.all;
entity shift is
   port(a,clk:in std_logic;
        b:out std_logic);
end shift;
architecture gen_shift of shift is
   component dff
      port(d,clk:in std_logic;
           g:out std_logic);
   end component;
   signal z:std_logic_vector(0 to 4);
begin
   a(0)<=a;  b<=z(4);
   dff1:dff port map(z(0), clk, z(1));
   dff2:dff port map(z(1), clk, z(2));
   dff3:dff port map(z(2), clk, z(3));
   dff4:dff port map(z(3), clk, z(4));
end gen_shift;
```

2、If --- generate 语句

语法：

```
标号：if  条件表达式  generate
            { 并行语句 }
        end  generate [标号];
```

if 语句与 If --- generate 的区别：

1、If --- generate 没有类似于 if 语句的 else 或 elsif 分支语句。

2、if 语句是顺序语句，If --- generate为并行语句。

- 例：

```
signal x: bit_vector(7 downto 0);
signal y: bit_vector(15 downto 0);
signal z: bit_vector(7 downto 0);
.....
G1: for i in x'range GENERATE
      z(i)<=x(i) AND y(i+8);
END GENERATE;
```

等效于：

z(7)<=x(7) AND y(15);
z(6)<=x(6) AND y(14);
.....
z(0)<=x(0) AND y(8);

并发执行！

注意1：

- GENERATE中循环操作的上界和下界必须是静态的。如为非静态参数，则往往不可综合。
- 例：

```
signal choice: integer range 0 to 3;

NotOK: FOR i IN 0 TO choice GENERATE
          (并发描述语句)
END GENERATE;
```

注意2：

- GENERATE语句使用过程中的多值驱动问题。
- 正确的用法：

```
OK: FOR i IN 0 TO 7 GENERATE
 output(i) <='1' when (a(i) AND b(i))='1' ELSE '0';
END GENERATE;
```

- 错误的用法：

```
NotOK: FOR i IN 0 TO 7 GENERATE
      accum<= "1111" when (a(i)='1') else "0000";
END GENERATE;
```

- 在顺序描述语句LOOP中则不存在多值驱动问题。

例：矢量移位器—使用GENERATE语句

功能描述：

- 输出矢量位宽是输入矢量的两倍；
- 输出矢量是输入矢量进行移位的结果，移位的次数由一个输入信号指定。
- 输入值为“1111”，输出值为下列值之一：

  row(0):00001111　--与输入相比，没有移位；

  row(1):00011110

  row(2):00111100

  row(3):01111000

  row(4):11110000

- 实现代码：

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity shifter is
    port ( inp: IN std_logic_vector(3 downto 0);
           sel: IN integer range 0 to 4;
           outp: OUT std_logic_vector(7 downto 0));
 end shifter;
architecture shifter of shifter is
    subtype vector is std_logic_vector(7 downto 0);
    type matrix is array(4 downto 0) of vector;
    signal row: matrix;
    begin
        row(0)<= “0000” & inp;
        G1: FOR i IN 1 TO 4 GENERATE
              row(i)<=row(i-1)(6 downto 0) & ‘0’;
        end GENERATE;
        outp<=row(sel);
    end
end shifter;
```

5.5 块（block）语句

块语句将一系列并行描述语句进行组合，目的是改善并行语句及其结构的可读性。可使结构体层次鲜明，结构明确。

两种类型的块：简单块（simple block）和卫氏块（guarded block）。

- 简单块：

仅仅是一种对原有代码进行区域分割的方式。将一系列的并发描述语句放在一个简单块中的目的是增强代码的可读性和可维护性。

语法结构：

```
标记：block
        { 块说明项 }
      begin
          { 并行语句 }
      end   block [ 标记 ];
```

- 使用简单块对一段构造体代码进行规整

```
architecture example  ...
begin
    ....
    block1: block
    begin
        ....
    end block block1;
    ....
    block2: block
    begin
        ....
    end block block2;
    ....
end example;
```

注意：块语句的使用不影响逻辑功能

以下两种描述结果相同：

描述一：

```
a1: out1<='1'  after 2 ns;
a2: out2<='1'  after 2 ns;
a3: out3<='1'  after 2 ns;
```

描述二：

```
a1: out1<='1'  after 2 ns;
blk1: block
   begin
      a2: out2<='1'  after 2 ns;
      a3: out3<='1'  after 2 ns;
   end  block blk1;
```

- 卫式（Guarded）块

与simple block相比，多了一个卫氏表达式。只有当卫氏表达式的值为真时，含有关键字guarded的语句才能执行（相当于条件执行语句，容易生成latch，小心！）。

语法结构：

```
标记：block （卫氏表达式）
        { 声明部分 }
      begin
          { guarded语句和其它并发描述语句 }
      end  block [ 标记 ];
```

- 例子1： 用guarded block实现锁存器

功能描述： 只有当clk='1'时，才执行语句q<=d;

```
LIBRARY ieee;
  USE  ieee.std_logic_1164.all;
  entity latch is
     port(d,clk:IN std_logic;
          q:OUT std_logic);
  end latch;
  architecture latch of latch is
  begin
     b1:block (clk='1')
     begin
        q<=guarded d;
     end block b1;
   end latch;
```

卫氏表达式

卫氏语句

- 例子2： 用guarded block实现D触发器

功能描述： 只有当clk为上升沿时，才执行卫氏语句;

```
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 dff of dff is
  begin
     b1：block (clk'EVENT AND clk='1')
     begin
        q<=guarded '0' when rst='1' else d;
     end block b1;
  end dff;
```

卫氏表达式

卫氏语句

- 注意：

  无论是simple block还是guarded block，内部都可以嵌套其他block，语法结构如下：

```
lable1: block
    [顶层block的声明部分]
begin
    [顶层block的并发描述语句]
            label2: block
                [嵌套block的声明部分]
            begin
                [嵌套block的并发描述语句]
            end block label2;
    [顶层block的其它并发描述语句]
 end block label1;
```

使用并发代码设计纯组合逻辑要注意一点：

不能有反馈信号！！！

- 如：

  b:=a NAND c;

  c:=b XOR d;

时序电路中反馈信号则很常见!

小结：

在并发代码中可以使用的项：

- WHEN语句（WHEN/ELSE语句或者WITH/SELECT/WHEN）
- GENERATE语句
- BLOCK语句
- 使用如逻辑、算术等运算操作符等的赋值语句也属于并发代码，可产生组合逻辑电路

第5章　思考题

1、块语句的作用是什么？有什么特点？

2、when/else语句与with/select/when语句在用法上有何区别？

3、生成（generate）语句与循环(loop)语句的异同点是什么？

实验课二：

（1）用GENERIC语句改写例4.1，设计成通用译码器，要求书写tb代码并仿出波形；

（2）课后习题5.6，要求书写tb代码并仿出波形；

（3）课后习题6.1，要求书写tb代码并仿出波形；