第6章 顺序代码

VHDL本质上是一种并发执行的代码，但是出于设计同步时序电路的需要，需要使用一些能够顺序执行的语句块，包括：PROCESS、FUNCTION、PROCEDURE。 这些语句块之间仍然是并发执行的，但内部是顺序执行的，称为顺序代码，又称行为描述代码。

使用顺序代码不但可以实现时序逻辑，还可以实现组合逻辑。

6.1、进程（process）

进程（process）内部的语句是一种顺序描述语句，其内部经常包括if,wait,case或loop语句。

特点：

1、进程与进程，或其它并发语句之间的并发性；

2、进程内部的顺序性；

3、要么使用敏感信号列表（sensitivity list），要么使用wait语句，二者不可同时使用。

4、进程必须包含在主代码段中，当敏感信号列表中的某个信号发生变化，或者wait语句的条件满足时，process内部的代码就顺序执行一次；

语法结构：

```
[标记：] process [( 敏感信号表)]
        [variable name: type[range][:=初始值]]
    begin
        { 顺序描述语句}
    end  process [标记];
```

可选，
增强代码可读性，
命名规则：非关键字

对临时连线的赋值,不可综合

•敏感信号表：进程内要读取的所有敏感信号（包括端口）的列表。每一个敏感信号的变化，都将启动进程。

格式：

信号名称{，信号名称}

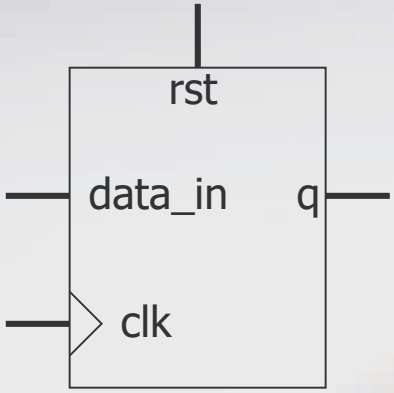
•注意：若电路模块对某输入信号不“敏感”，则该输入信号不必放入敏感信号列表中。

•如同步时序电路中只对clk和rst这两个输入信号敏感，而对data,addr等输入信号不敏感，则敏感信号列表中只需放入clk和rst即可；

•在纯组合逻辑电路中，电路模块对任意一个输入信号都是敏感的，所以如果要使用process来实现一段组合逻辑，则必须将所有的输入信号都放入敏感信号列表中。

- 在使用顺序代码实现一个同步时序电路时，必须对某些信号边沿的跳变进行监视（典型的是时钟信号clock的上升沿或下降沿）
- 通常使用EVENT来监视一个信号是否发生了边沿跳变
- 通常在process中使用敏感信号clk来实现同步时序电路。

例1：带有异步复位端的D触发器



功能描述： **时序逻辑电路的基本单元**

- **当输入的时钟信号为上升沿时，输出信号q等于当前输入值；**
- **异步复位端rst（reset）：rst=‘1’时q=‘0’；优先级高于当前输入值；**

- 实现代码

```
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 behavior of dff is
begin
    process(clk,rst)
    begin
        if (rst='1') then
            q<=0;
        else (clk'event AND clk='1') then
            q<=d;
        end if;
     end process;
end behavior;
```

任意一个信号发生变化时，process中的所有语句就执行一次

敏感信号表的特点：

1、同步进程的敏感信号表中只有时钟信号。

如：

```
process(clk)
begin
    if(clk'event and clk = '1')  then
        if  reset = '1'  then
            data <= "00";
        else
            data <= in_data;
        end  if;
    end  if;
end  process;
```

2、异步进程敏感信号表中除时钟信号外，还有其它信号。

例：

```
process(clk, reset)
begin
    if   reset = '1'  then
         data <= "00";
    elsif(clk'event and clk = '1') then
         data <= in_data;
    end  if;
end  process;
```

3、如果有 wait 语句，则不允许有敏感信号表。

```
PROCESS (a,b)
BEGIN
    --sequential statements
END PROCESS;
```

```
PROCESS
BEGIN
    WAIT ON/Until signalx/signal_expr ;  //先
    -- sequential statements    //后
END PROCESS;
```

6.2 信号和变量的基本知识

VHDL中两种动态的传递数值的方法：信号与变量。

两者的差异：

- 有效范围的不同：

  信号：程序包、实体、结构体；全局量。

  变量：进程、子程序；局部量。

- 赋值方式的不同：

  变量：= 表达式；

  信号 <= 表达式；

- 赋值行为的不同：

  信号赋值延迟更新数值、时序电路；

  变量赋值立即更新数值、组合电路。

6.3 IF语句

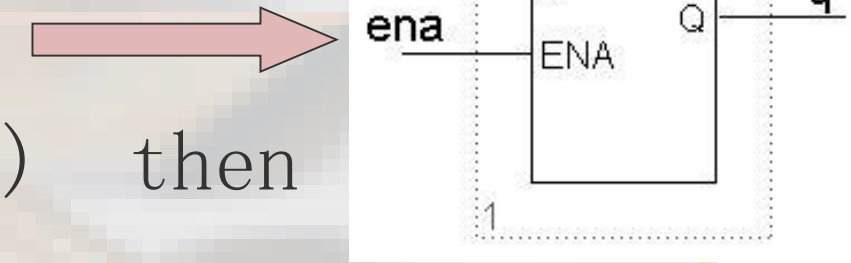
只能用于顺序代码，只能在process、function和procedure中出现。

语法结构：

```
if (表达式) then
   语句1;    --可以多个语句，不必使用{}
   语句2;
   …
elsif (表达式) then    或者   else
  …
end if;
```

1）if 语句的门闩控制

```
if  条件  then
    顺序处理语句;
end if ;
```



例：if (ena = ‘1’) then

q <= d;

end if; ---没有else描述！

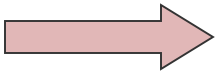
综合后生成锁存器（latch）

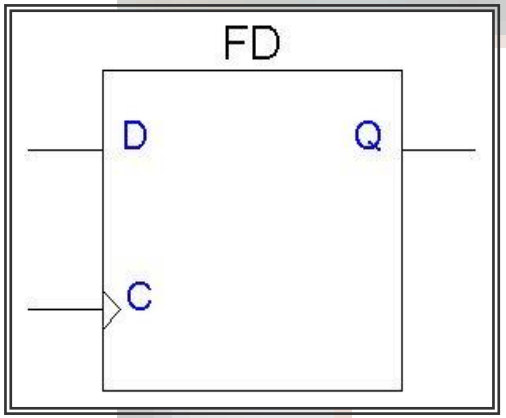
注意：用if或者case语句做逻辑电路的时候，必须为信号设置默认值，避免生成latch电路。有两种方法：

- 在if，case语句之前对目标信号进行赋值，采用这种方法，就不必专门写else或者when others语句对信号进行默认赋值。
- 在else或者when others语句中对信号进行默认条件下的赋值。
- 如果违反了上述规则，那么会在综合电路的时候形成一个transparent latch，也就是电平触发的锁存器，这对电路的时序分析等会造成很大的麻烦。

条件改为时钟沿，则生成 D触发器：

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





注意：在时序电路中，如果没有在else语句或者when others语句中对信号赋值，那么综合工具会认为寄存器保持当前输入。

2）if 语句的二选择控制

格式：

```
if  条件  then
    顺序处理语句;
else
    顺序处理语句;
end  if ;
```

用条件来选择两条不同程序执行的路径。

3）if 语句的多选择控制

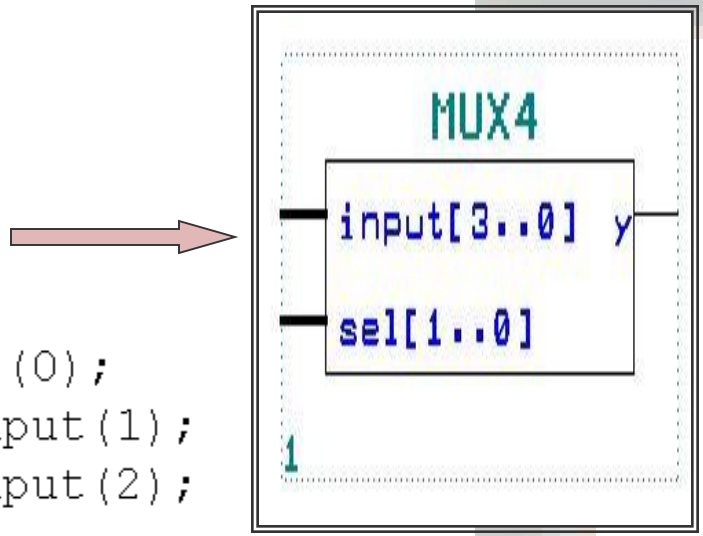
if 语句的多选择控制又称为 if 语句的嵌套。

格式：

```
if  条件  then
    顺序处理语句;
elsif  条件  then
    顺序处理语句;
      ⋮
elsif  条件  then
    顺序处理语句;
else
    顺序处理语句;
end  if;
```

典型电路是多选一(四选一)电路。

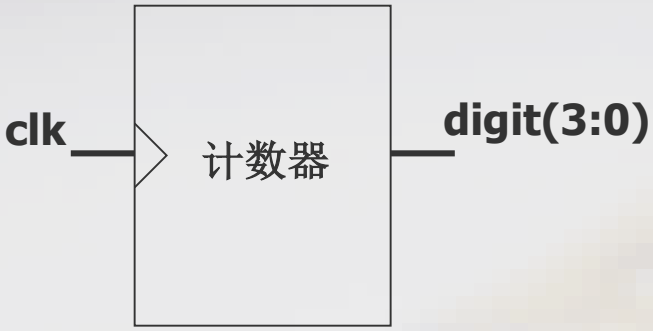
```
library ieee;
use ieee.std_logic_1164.all;
entity mux4 is
   port (input:in std_logic_vector(3 downto 0);
         sel:in std_logic_vector(1 downto 0);
         y:out std_logic);
end mux4;
architecture rtl of mux4 is
begin
   process(input,sel)
   begin
      if (sel="00") then y<=input(0);
      elsif (sel="01") then y<=input(1);
      elsif (sel="10") then y<=input(2);
      else y<=input(3);
      end if;
   end process;
end rtl;
```



if_then_elsif 语句中隐含了优先级别的判断，最先出现的条件优先级最高，可用于设计具有优先级的电路，如8-3优先级编码器。

如果不希望出现优先级的情况，则可使用CASE语句。

例6.2 模10计数器



功能描述：循环累加clk上升沿的次数，输出到4位的digit。

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY counter IS
    port ( clk: IN std_logic;
           digit: OUT integer range 0 to 9);
END counter;
```

```
ARCHITECTURE counter of counter IS
begin
    count: process(clk)
        variable temp: integer range 0 to 10;
    begin
        if (clk'event AND clk='1') then
            temp:=temp+1;
            if (temp=10) then temp:=0;
            end if;
        end if;
        digit<=temp;
    end process count;
end counter;
```

模10计数时，该段代码有哪些缺陷？有奖竞答！

```
ARCHITECTURE counter of counter IS
begin
    count: process(clk)
        variable temp: integer range 0 to 10;
    begin
        if (clk'event AND clk='1') then
            temp:=temp+1;
            if (temp=10) then temp:=0;
            end if;
        end if;
        digit<=temp;
    end process count;
end counter;
```

4bit位宽

未赋初值，因为4位宽，所以可能>10而<=15:周期的浪费！
另，不能从0开始计数！可加复位信号

不是latch，而是二选一的选择器

常数比较电路，代价较大，如何优化？
与常数0进行比较，递减计数，一个简单的“与”即可，不必构建复杂的全比较器！

```
ARCHITECTURE counter of counter IS
begin
    count: process(clk)
        variable temp: integer range 0 to 10;
    begin
if (clk'event AND clk='1') then
            temp:=temp-1;
            if (temp=0) then temp:=10;
            end if;
        end if;
        digit<=temp;
    end process count;
end counter;
```

6.4 WAIT语句

- 用法与IF语句类似，且形式更加多样。
- 如果在process内部使用了wait语句，则process就不能再使用敏感信号列表。并且wait语句是process内部的第一条语句。
- wait条件满足时，process内部代码就执行一次。
- 常用的三种语法：

```
wait   on   单个或多个信号;        -- 敏感信号量变化
wait   until 单个信号条件;         -- 条件满足（可综合）
wait   for   time;                 -- 时间到（只能仿真，
                                      不可综合）
```

例6.5 使用wait until语句设计模10计数器

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity counter is
   port (clk: IN std_logic;
         digit: OUT integer range 0 to 9);
end counter;
architecture counter of counter is
begin
   process   ---没有敏感信号列表
      variable temp: integer range 0 to 10;
    begin
       wait until (clk'event AND clk='1');
        temp:=temp+1;
        if (temp=10) then temp:=0;
        end if;
         digit<=temp;
     end process;
end counter;
```

第一条

6.5 CASE语句

语法结构：

```
CASE 表达式 IS
        WHEN 条件表达式=>顺序执行语句；
        WHEN 条件表达式=>顺序执行语句；
       ……
  END  CASE；
```

注意：在每个测试条件下case语句允许执行多个赋值操作，而with/select语句只允许执行一个赋值操作。

例：

```
CASE control IS
  when “00”=>x<=a; y<=b;
  when “01”=>x<=b; y<=c;
  when OTHERS=>x<=“0000”;y<=“zzzz”;
END CASE;
```

代表了所有未列出的可能情况。

注意： 可使用关键字NULL来表示操作没有发生，如：

```
when others=>null;
```

6.6 LOOP语句

- 当一段代码需要多次重复、顺序地执行时，loop语句非常有效。
- 只能在process、function和procedure中使用；
- 与generate语句的最大不同：

generate语句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，与先后顺序无关；

loop语句则按顺序、循环地执行某项操作。

LOOP语句的语法结构：

•FOR/LOOP：循环固定次数

上/下界必须是静态值，有一个计数比较模块来加以控制

[label:] FOR 循环变量 IN 范围 LOOP

（顺序描述语句）

END LOOP [label]；

•WHILE/LOOP：循环执行直到某个条件不再满足

[label:] WHILE 条件表达式 LOOP

（顺序描述语句）

END LOOP [label]；

•EXIT：结束整个循环操作

[label:] EXIT [label] [WHEN 条件表达式];

•NEXT：跳出本次循环操作

[label:] NEXT [loop_label] [WHEN 条件表达式];

例：

```
FOR i IN 0 TO data'range LOOP
   case data(i) is
      when '0' =>count:=count+1;
       when others=> EXIT;
    end case;
END loop;
```

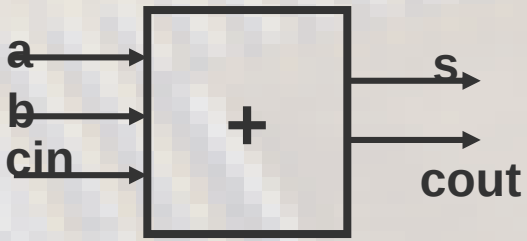
例6.8 逐级进位加法器（vs 并行进位加法器）

功能描述：最为常用的基本功能模块。

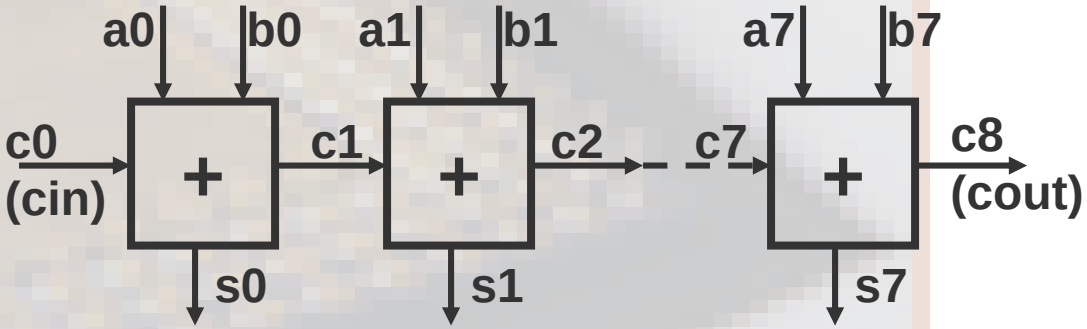
•输入矢量a和b为8位无符号数，cin为输入的进位位，s为输出的和，cout为输出的进位位；

$s_j = a_j \text{ XOR } b_j \text{ XOR } c_j$;

$c_{j+1} = (a_j \text{ AND } b_j) \text{ OR } (a_j \text{ AND } c_j) \text{ OR } (b_j \text{ AND } c_j)$;



一位全加器



8位逐级进位加法器

代码的实现方案一：----使用generic语句

```
library ieee;
use ieee.std_logic_1164.all;
entity  adder is
     generic (length: integer : =8);
     port (a, b: IN std_logic_vector (length-1 donwto 0);
              cin: IN std_logic;
               s: OUT std_logic_vector (length-1 downto 0);
               cout: OUT std_logic );
end adder;
architecture adder of adder is
begin
    process (a, b, cin)
        variable carry: std_logic_vector (length downto 0);
    begin
         carry(0) : =  cin;
        for i IN 0 TO length-1 LOOP
              s(i)<= a(i) XOR b(i) XOR carry(i);
              carry(i+1):= (a(i) AND b(i) ) OR (a(i) AND carry(i)) OR (b(i) AND carry(i));
         end LOOP;
    end process;
end adder;
```

代码的实现方案二： ----不使用generic语句

```
library ieee;
use ieee.std_logic_1164.all;
entity  adder is
      port (a, b: IN integer range 0 to 255;
            c0: IN std_logic;
             s: OUT integer range 0 to 255;
             c8: OUT std_logic );
end adder;
architecture adder of adder is
begin
    process (a, b, c0)
        variable temp: integer range 0 to 511;
    begin
        if (c0='1') then temp:=1;
        else temp:=0;
        end if;
       temp:=a+b+temp;
       if (temp>255) then
          c8<='1';
          temp:=temp-256;
        else c8<='0';
        end if;
        s<=temp;
    end process;
end adder;
```

std_logic_vector类型，不能直接temp：=c0；

integer类型，可直接进行算术运算；

Temp范围>s范围

6.7 CASE语句和IF语句的比较

原则上说来，IF/ELSE语句中ELSE的出现可能会造成综合后的电路中出现优先级解码器。（CASE语句中不会出现这种情况），但在实际综合时综合工具的优化功能往往不会出现这种情况。因此，使用IF语句和CASE语句编写的代码在综合优化后最终生成的电路结构通常是一样的。 注意编码风格！

例子：

----------IF语句----------

```
IF (sel="00") then x<=a;
elsif (sel="01") then x<=b;
elsif (sel="10") then x<=c;
else x<=d;
END IF;
```

----------CASE语句----------

```
CASE  sel   IS
   when "00"=>x<=a;
   when "01"=>x<=b;
    when "10"=>x<=c;
    when others=>x<=d;
end case;
```

6.8 CASE语句和WITH语句的比较

	WITH（SELECT/WHEN）	CASE（WHEN）
代码类型	并发代码	顺序代码
用法	在PROCESS、FUNCTION和PROCEDURE外部使用	在PROCESS、FUNCTION和PROCEDURE内部使用
是否必须列出所有可能的组合	是	是
每个判断分支允许的最多赋值语句数目	1	任意数目
没有操作动作时使用的关键字	UNAFFECTED	NULL

例：使用with语句和case语句

```
----------使用with语句----------
WITH sel SELECT
      x<=a when “00”;
           b  when “01”;
           c   when “10”;
            unaffected  when OTHERS;
----------使用case语句----------
CASE sel IS
      when “00”=>x<=a;
      when “01”=>x<=b;
      when “10”=>x<=c;
      when OTHERS=>null;
 END CASE;
```

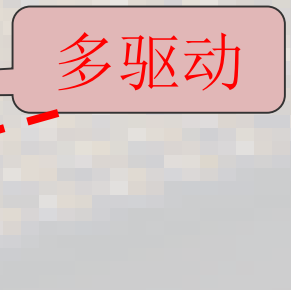
6.9 同步时序电路中的时钟问题

A: 如果在参考时钟的两个边沿（上升沿和下降沿）都触发对同一个信号的赋值操作，那么这样的代码通常是不可综合的。

如果出现这种情况，编译器将提示“信号值在时钟边沿不能保持（hold）”等警告信息。

例子：一个在每个时钟沿都进行计数操作的计数器

```
process (clk)
begin
   if(clk'event AND clk='1') then
        counter<=counter+1;
   elsif(clk'event AND clk='0') then
        counter<=counter+1;
   end if;
end process;
```



（注意：“在参考时钟的两个边沿（上升沿和下降沿）都触发对同一个信号的赋值操作通常不可综合”不是绝对的，通过特殊的设计技术，在一些专用的、高速电路中是可以综合的，如DDR DRAM）

B：EVENT属性必须与某个测试条件关联，否则容易存在二义性。

例如语句IF（clk'event）就存在二义性，可能是由高电平变为低电平，也可能是由低电平变为高电平。对该语句的解释因编译器的不同而不同。有的编译器认为AND clk='1'是默认的，而有的编译器则会警告“时钟不稳定”。 例：

```
process(clk)
begin
  if(clk'event) then counter:=counter+1;
  end if;
end process;
```

存在二义性的语句不被执行！

C：如果一个信号出现在process的敏感信号列表中，但没有在process内部的任何语句中出现，则编译器往往将其忽略。

例：

```
process(clk)
begin
    counter:=counter+1;
    ....
end process;
```

虽然看起来“只要clk发生变化，process内的语句就顺序执行一次”，但是由于clk未在process内部出现，编译器将提示“忽略了不必要的信号clk”，未达到设计的初衷。

D：“在参考时钟的两个边沿（上升沿和下降沿）都触发对同一个信号的赋值操作通常不可综合”=>“在参考时钟的两个边沿（上升沿和下降沿）分别触发信号a、b的赋值操作则是可以综合的”。

例：

```
process(clk)
begin
    if (clk'event AND clk='1') then
        x<=d;
    end if;
end process;

process(clk)
begin
    if (clk'event AND clk='0') then
        y<=d;
    end if;
end process;
```

例6.11 双数据总线的RAM设计

- 功能描述：

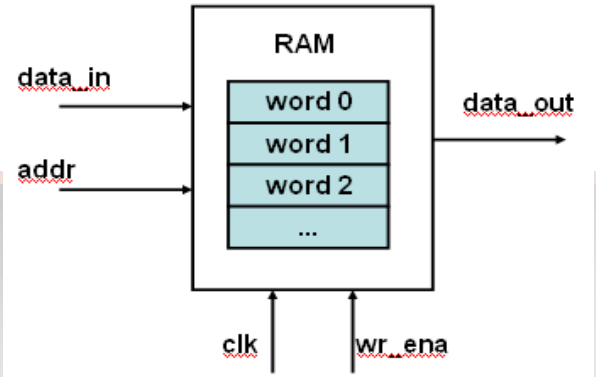
在输入方向，当写使能信号wr_ena有效时（高电平），在随后的一个时钟上升沿出现时，输入数据总线上的数据将被写入地址总线所指定的位置上；当写使能信号wr_ena无效时（低电平），不会有新数据被写入RAM中。

在输出方向，输出数据总线上显示的数据始终是当前地址总线所指定的RAM空间中的内容。

- 端口定义： 分离的输入/输出数据总线（data_in、data_out）、clk、wr_ena（高电平有效，若低电平有效，则往往定义为nWR_ENA）、addr。

- 规格： RAM位宽为8bit，深度为16；使用GENERIC语句设计通用代码；

- 设计要点：IF语句




```
library ieee;
use ieee.std_logic_1164.all;
entity ram is
generic (   bits: integer:=8;  --位宽
            words:integer:=16;  --深度   );
port (wr_ena, clk:IN std_logic;  addr: IN integer range 0 to words-1;
      data_in: IN std_logic_vector (bits-1 downto 0);
      data_out: OUT std_logic_vector (bits-1 downto 0) );
end ram;
architecture ram of ram is
   type vector_array is array(0 to words-1) of std_logic_vector(bits-1 downto 0);
   signal memory: vector_array;
begin
    process(clk, wr_ena)
    begin
        if (wr_ena='1') then
            if (clk'event and clk='1') then
                memory(addr)<=data_in;
            end if;
        end if;
    end process;
    data_out<=memory(addr);
end ram;
```

独立于process，不受clk与wr_ena的影响

6.10 使用顺序代码设计组合逻辑电路

- 需遵循的原则：

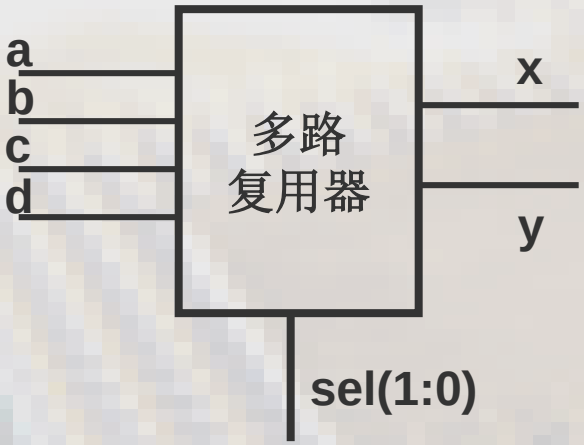
①确保在process中用到的所有输入信号都出现在敏感信号列表中。（否则，编译器提示warning：“...”）

②确保考虑了输入/输出信号的所有可能组合，即电路的真值表必须在代码中完整地反映出来（并发代码与顺序代码都一样），多使用others。（否则，生成latch）

例6.12：会错误生成latch的组合逻辑设计

功能描述：实现一个多路复用器。

- x是输出端之一，输出值等于sel所选择的一个输入端的值；
- y是输出端之二，当sel="00"时，y='0';当sel="01"时，y='1';



sel	x	y
00	a	0
01	b	1
10	c	
11	d	

完全依据文字描述得出的真值表

实现代码

```
library ieee;
use ieee.std_logic_1164.all;
entity  example is
     port (a, b, c, d: IN std_logic;
           sel: IN integer range 0 to 3;
           x, y: OUT std_logic );
end example;
architecture example of example is
begin
    process (a, b, c, d, sel)
    begin
        if (sel=0)    then       x<=a;        y<='0';
        elsif (sel=1)  then    x<=b;        y<='1';
        elsif  (sel=2)   then  x<=c;
        else                          x<=d;
        end if;
    end process;
end example;
```

y的赋值？

•仿真结果：当sel=2或3时，y的波形异常，同一输入，不同输出结果；

•综合结果：有latch生成。

sel	x	y
00	a	0
01	b	1
10	c	y
11	d	y

代码所实际实现的真值表

•原因分析：文字描述中还有一层隐含的意思未被表达出来：

当sel=“10”或“11”时，无需考虑y的值；

•正确的代码实现方法： 在使用条件语句如if或case 时，如果遇到“无需考虑某个条件下某信号的赋值”或“某信号的值不能确定”时，务必要将std_logic类型的值‘x’赋值给该信号。

sel	x	y
00	a	0
01	b	1
10	c	X
11	d	X

正确实现的真值表

```
.................
process (a, b, c, d, sel)
    begin
        if (sel=0)   then      x<=a;     y<='0';
        elsif (sel=1) then     x<=b;     y<='1';
        elsif  (sel=2)  then   x<=c;     y<='x';
        else                   x<=d;     y<='x';
        end if;
.................
```

- 语法记忆点：
- process (clk,..)　---无process_name;

  ……

  end process;
- if (…) then　　---无if_name;

  end if;
- Entity entity_name IS　　---有entity_name;

  …

  end entity_name;
- Architecture arch_name of entity_name IS　---有arch_name;

  …..

  end arch_name;

第6章 课后思考题与作业

- 顺序代码有哪些标志性语句？--process...
- process语句的敏感信号列表特点？
- if语句使用不当会带来哪些负面影响？

  1、latch

  2、优先级编码电路
- case语句与with/select语句的异同？
- 在同步时序电路中，能否在两个clk边沿对同一个信号赋值？