第7章 信号(Signal)和变量(Variable)

- VHDL处理静态数据的两种对象：const和generic。
- VHDL处理非静态数据的两种对象：信号和变量。
- 常量和信号是全局的，既可以用于顺序代码，也可用于并发代码；
- 变量只能在顺序代码中使用，相对于信号而言，变量只能是局部的，所以变量值不能传递到process、function和procedure外部。
- 但是，在有些情况下，选择信号还是变量却是比较难以抉择的。

7.1 常量

用于确定默认值，语法结构：

CONST 常量名：type：=值；

例子： CONST set_bit : BIT:='1';

CONST datamemory：memory:=(('0','0','1','1'), ('0','0','1','1'));

常量可以在包集、实体或结构体中声明：

- 包集中：全局；
- 实体中：对该实体内的所有结构体而言是全局的；
- 结构体中：结构体内是全局的。

7.2 信号

VHDL中‘信号’代表电路单元、功能模块间的硬件连线，也可表示电路单元的IN/OUT端口；

实体的所有端口都默认为信号，语法结构：

Signal 信号名：type [range][：=初始值]；

注意：当信号用于顺序描述语句如process中时，其值不是立刻更新的；只有当其所在的process、函数或过程完成之后才进行更新。

信号的赋值符号： <=

注意1：对信号赋初值的操作是不可综合的，通常只用于仿真。

例如： signal control: BIT:='0';

注意2：不要对同一个信号进行多重赋值。（buffer模式的端口信号除外！）

例如：

```
process (...)
  .....
  for i IN 0 TO 10 LOOP
    control<=control+1;      --error! 或最后一次有效！
  ......
```

例7.1 “1”计数器

- **功能描述：** 计算一个二进制矢量中‘1’的个数

- 实现代码：---错误使用信号的例子

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity count_ones is
   port ( din: IN std_logic_vector(7 downto 0);
          ones: OUT integer range 0 to 8);
 end count_ones;
architecture not_ok of count_ones is
    signal temp: integer range 0 to 8;
begin
    process (din)
    begin
       temp<=0;
       FOR i IN 0 TO 7 LOOP
           IF (din(i) ='1') then temp<=temp+1;
           END if;
       END LOOP;
       ones<=temp;
   end  process;
end not_ok;
```

进程结束后才更新

两种代码更正方法：

1、使用变量来记录中间值，可以将signal temp...改为variable temp...；同时修改相应的赋值语句；

2、取消temp，同时将端口信号ones重新定义为：

ones: **BUFFER** integer range 0 to 8;

使得ones可以被内部调用。在进程结束后，ones值将被更新。（编码风格不好，不提倡此种方式！可以参考P110-P111例子）

7.3 变量(variable)

•变量代表电路单元内部的操作，代表暂存的临时数据。与信号和常量相比，变量仅用于局部的电路描述，只能用于进程、函数和过程内部。

•注意：对变量的赋值是立即生效的，无需等待进程结束。新的值可以在下一行代码中立即使用。

•变量的赋值符号“：=”，语法结构：

```
variable 变量名：type [range][：=初始值];
```

注意：对变量赋初值的操作也是不可综合的，通常只用于仿真。

例：“1”计数器的实现代码：

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity count_ones is
   port ( din: IN std_logic_vector(7 downto 0);
          ones: OUT integer range 0 to 8);
end count_ones;
architecture ok of count_ones is
    variable temp: integer range 0 to 8;
begin
    process (din)
    begin
        temp：=0;
        FOR i IN 0 TO 7 LOOP
            IF (din(i) =‘1’) then temp：=temp+1;
            END if;
        END LOOP;
        ones<=temp;
    end  process;
end ok;
```

7.4 信号和变量的比较 （补充）

1）赋值方式的不同：

变量：= 表达式；

信号 < = 表达式；

2）硬件实现的功能不同：

信号代表电路单元、功能模块间的互联，代表实际的硬件连线；

变量代表电路单元内部的操作，代表暂存的临时数据。

3）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。

ARCHITECTURE

{**SIGNAL** Declarations}

label1: PROCESS
{**VARIABLE** Declarations}

⋮

label2: PROCESS
{**VARIABLE** Declarations}

4）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。

5）信号的多次赋值

a. 一个进程：最后一次赋值有效

b. 多个进程：多源驱动

线与、线或、三态

例：信号的多次赋值 （补充）

```
architecture rtl of ex is
    signal  a : std_logic;
begin
    process(…)
    begin
        a <= b;
          …
        a <= c;
    end  process;
end rtl;
```

```
architecture rtl of ex is
    signal  a : std_logic;
begin
    process(…)
    begin
        a <= b;
          …
    end  process;

    process(…)
    begin
        a <= c;
          ...
     end  process;
end ex;
```

例：信号赋值与变量赋值的比较　（补充）

信号赋值：

```
architecture  rtl of sig  is
signal a,b : std_logic;  -- 定义信号
begin
  process(a, b)
  begin
     a <= b ;
     b <= a ;
  end  process ;
end  rtl ;   -- 结果是 a 和 b 的值互换
```

变量赋值：　（补充）

```
architecture  rtl  of  var  is
begin
process
   variable a,b:std_logic;    -- 定义变量
   begin
       a := b ;
       b := a ;
           end  process ;
end rtl;
```

-- 结果是a和b的值都等于b的初值

例：变量赋值实现循环语句功能　（补充）

```
process(indicator, sig)
    variable  temp : std_logic;
begin
    temp := '0' ;
    for  i  in  0  to  3  loop
        temp:=temp xor (sig(i) and indicator(i));
    end  loop ;
    output <= temp;
end  process;
```

以上语句等效为：

```
process(indicator, sig)
  variable  temp : std_logic ;
begin
  temp := '0' ;
  temp :=temp xor (sig(0) and indicator(0));
  temp :=temp xor (sig(1) and indicator(1));
  temp :=temp xor (sig(2) and indicator(2));
  temp :=temp xor (sig(3) and indicator(3));
  output <= temp ;
end  process ;
```

如改为信号，则无法实现原功能：　（补充）

```
……
signal  temp : std_logic;
……
process(indicator, sig, temp)
begin
    temp<= ‘0’ ;
    temp<=temp xor (sig(0) and indicator(0));
    temp<=temp xor (sig(1) and indicator(1));
    temp<=temp xor (sig(2) and indicator(2));
    temp<=temp xor (sig(3) and indicator(3));
    output <= temp ;
end  process ;
```

例7.3：多路复用器的对比设计

方案一：使用信号（not OK）

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity mux is
   port ( a, b, c, d, s0, s1: IN std_logic;
          y: OUT std_logic);
 end mux;
architecture not_ok of mux is
    signal sel: integer range 0 to 3;
begin
    process (a, b, c, d, s0, s1)
    begin
        sel<=0;
        if (s0='1') then sel<=sel+1;
        end if;
        if (s1='1') then sel<=sel+2;
        end if;
       case sel is
            when 0=>y<=a;
            when 1=>y<=b;
            when 2=>y<=c;
            when 3=>y<=d;
       end case;
    end  process;
end not_ok;
```

值不能立即更新，不能在process的其它代码中继续使用

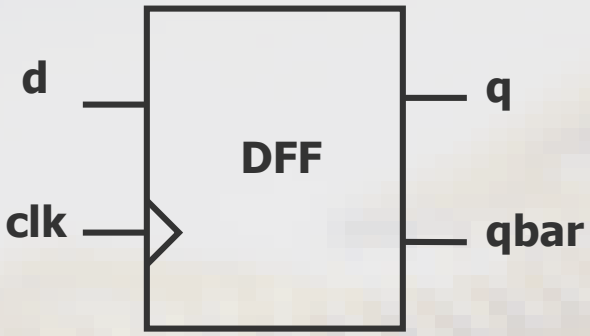
不能进行同一信号的多次赋值，要么出错，要么“线与”，要么只考虑最后一次赋值，取决于编译器

例7.3： 方案二：使用变量（OK）

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity mux is
   port ( a, b, c, d, s0, s1: IN std_logic;
          y: OUT std_logic);
 end mux;
architecture ok of mux is
begin
    process (a, b, c, d, s0, s1)
        variable sel: integer range 0 to 3;
    begin
        sel:=0;
        if (s0='1') then sel:=sel+1;
        end if;
        if (s1='1') then sel:=sel+2;
        end if;
       case sel is
            when 0=>y<=a;
            when 1=>y<=b;
            when 2=>y<=c;
            when 3=>y<=d;
       end case;
   end  process;
end  ok;
```

值可以立即更新，能够在process的其它代码中继续使用

例7.4 带q和qbar的DFF



功能描述： D触发器

qbar为q的反相输出端。

方案1： not_ok

```
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
entity dff is
   port( d, clk: IN std_logic;
         q: BUFFER std_logic;
         qbar: OUT std_logic );
    end dff;
-----------------------------------------------
architecture not_ok of dff is
 begin
    process (clk)
    begin
        if (clk'event AN     k='1')  then
            q<=d;
            qbar<=NOT q;
        end if;
    end process;
 end not_ok;
-----------------
```

q不能立即更新，导致qbar值将延后一个时钟周期

buffer类型输出信号可供内部电路使用

方案2： ok

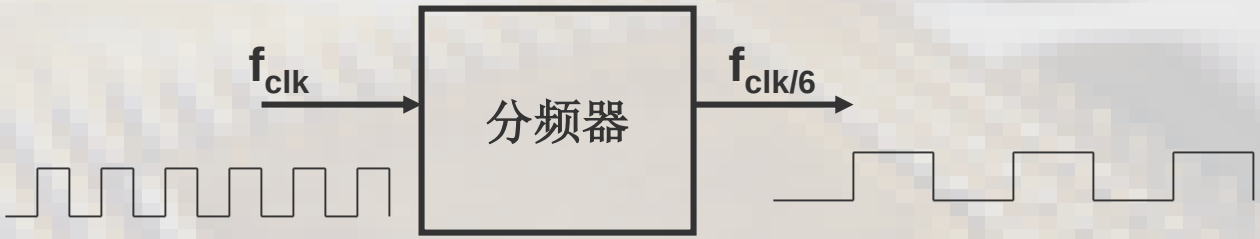
```
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
entity dff is
   port( d, clk: IN std_logic;
         q: BUFFER std_logic;
         qbar: OUT std_logic );
    end dff;
-----------------------------------------------
architecture ok of dff is
 begin
    process (clk)
    begin
        if (clk'event AN      ='1') then
            q<=d;
        end if;
    end process;
    qbar<=NOT q;
end mux2;
-----------------------------------
```

q值在process之后更新，引起qbar的同步更新

书上波形图

例7.5 分频器

- **功能描述**：对时钟进行6分频；
- **设计要点**：两个输出，一个基于**信号**，另一个基于**变量**。



实现代码：

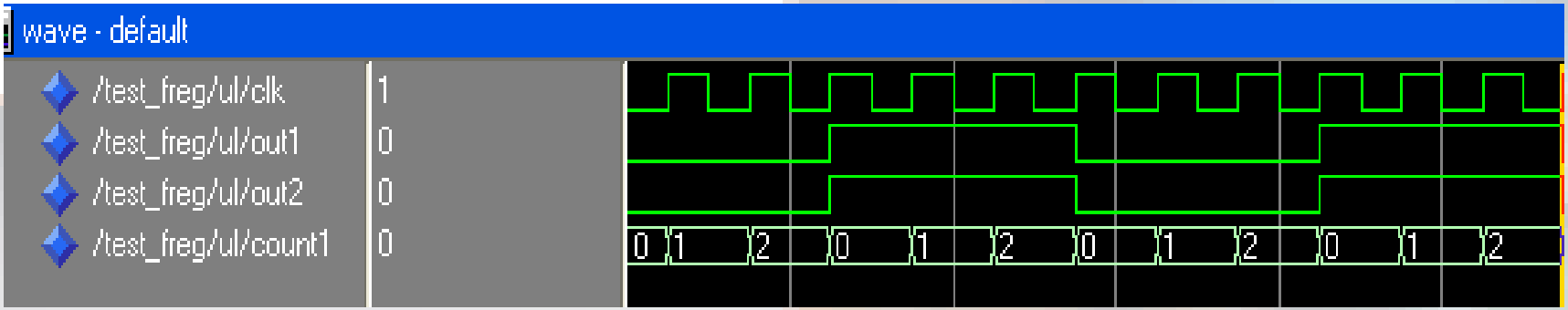
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
entity freq_divider is
   port ( clk: IN std_logic;
          out1, out2: BUFFER std_logic);
 end freq_divider;
architecture example of freq_divider is
        SIGNAL count1: integer range 0 to 7;
begin
    process (clk)
        variable count2: integer range 0 to 7;
    begin
        if (clk'event AND clk='1') then
            count1<=count1+1;
            count2:=count2+1;
                if (count1=___) then
                    out1<=NOT out1;
                    count1<=0;
                end if;
                if (count2=___) then
                    out2<=NOT out2;
                    count2:=0;
                end if;
            end if;
    end  process;
end  example;
```

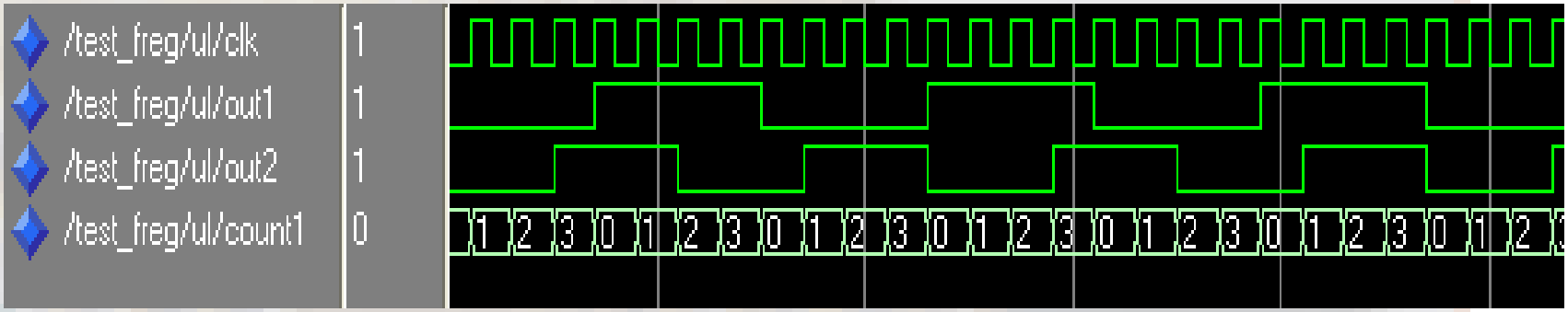
未赋初值，但范围固定

buffer类型，可以实现电平翻转；
另，将std_logic改为bit类型更好些!

计数清零

信号延迟更新，变量即时更新，填空区别？


wave - default
/test_freg/ul/clk
/test_freg/ul/out1
/test_freg/ul/out2
/test_freg/ul/count1
1
0
0
0
0 1 2 0 1 2 0 1 2 0 1 2


/test_freg/ul/clk
/test_freg/ul/out1
/test_freg/ul/out2
/test_freg/ul/count1
1
1
1
0
1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2

注意：

out1和out2都没有赋初值，仿真波形中 会出现在仿真之处波形不太对齐的问题，但由于是分频器，只要产生时钟信号即可，不必在意时钟信号的起点是高电平还是低电平！

7.5 寄存器的数量---对电路面积的重要影响

不同的编译器（仿真工具、综合工具等）对不同风格的代码进行编译时产生的寄存器数量是不同的。本小节目的：

- 了解可以使用什么方法来减少寄存器数量；
- 了解代码是否可以实现预期的结果；

寄存器生成的条件：

- 对信号来说：当一个信号的赋值是以另一个信号的边沿或跳变为条件时（即发生同步赋值时），该信号经编译后就会生成寄存器。同步赋值只能在进程、函数或过程等顺序代码中出现，一般在语句**if signal'event....**或**wait until**之后。
- 对变量来说： **(1)**若一个变量是在一个信号跳变时被赋值，并且该值最终又被赋给了另外的信号（数值传递作用），则将生成寄存器 。**(2)**一个变量未进行赋值操作时已经被信号使用，也将产生寄存器（用以存储**process**上一次执行后**c**的值，参见例**7.8**方案**1**）。
- 如果未被进程、函数或过程之外的代码调用，则不一定生成寄存器。

例：output1和output2都被寄存的情况

```
process (clk)
 begin
     if (clk'event AND clk='1') then
          output1<=temp;    --被存储或生成寄存器；
          output2<=a;          --被存储或生成寄存器；
      end if;
  end process;
```

原因：对信号来说：当一个信号的赋值是以另一个信号的边沿或跳变为条件时（即发生同步赋值时），该信号经编译后就会生成寄存器。同步赋值只能在进程、函数或过程等顺序代码中出现，一般在语句if signal'event....或wait until之后。

例：output1被寄存、output2未被寄存的情况

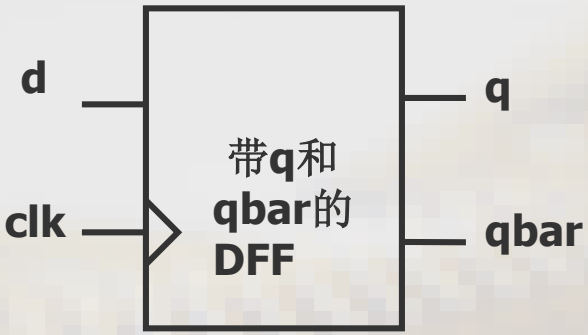
```
process (clk)
 begin
     if (clk'event AND clk='1') then
          output1<=temp;    --被存储或生成寄存器；
     end if;
     output2<=a;           --未以另一个信号的边沿跳变进行赋值，故未被存储；
      end if;
 end process;
```

例：变量被寄存的情况

```
process (clk)
        variable temp: BIT;
 begin
     if (clk'event AND clk='1') then
          temp:=a;
      end if;
          x<=temp;          --temp促使x被存储；
  end process;
```

原因：对变量来说： 若一个变量是在一个信号跳变时被赋值，并且该值最终又被赋给了另外的信号（数值传递作用），则将生成寄存器 。

例7.6：带q和qbar的DFF



功能描述： D触发器

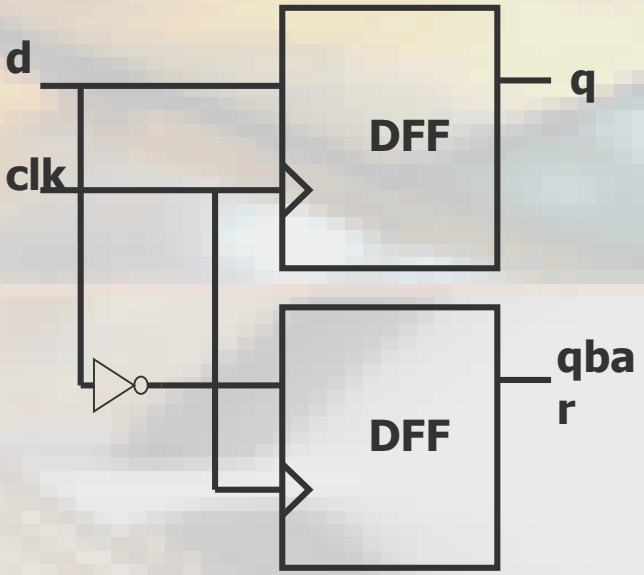
qbar为q的反相输出端。

两种实现方案：方案一生成两个寄存器，方案二生成一个寄存器。

方案1： two-dff

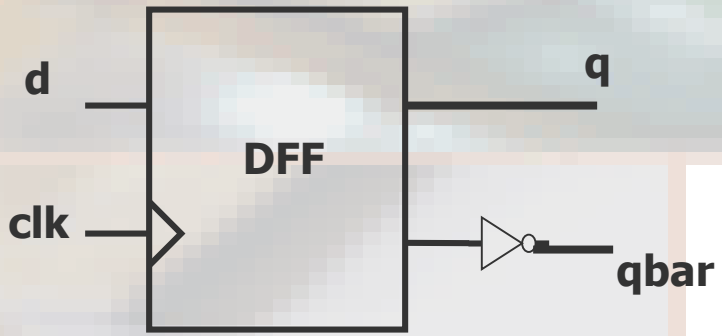
```
-------------------------------------------------
 library ieee;
 use ieee.std_logic_1164.all;
-------------------------------------------------
 entity dff is
    port( d, clk: IN std_logic;
          q: BUFFER std_logic;
          qbar: OUT std_logic );
     end dff;
-----------------------------------------
architecture two_dff of dff is
 begin
    process (clk)
    begin
        if (clk'event AN     k='1')  then
            q<=d;
            qbar<=NOT q;
        end if;
     end process;
 end two_dff;
 ------------------------------------
```

生成两个寄存器



方案2： one-dff

```
-------------------------------------------------
 library ieee;
 use ieee.std_logic_1164.all;
-------------------------------------------------
 entity dff is
    port( d, clk: IN std_logic;
          q: BUFFER std_logic;
          qbar: OUT std_logic );
     end dff;
-------------------------------------------
architecture one_dff of dff is
 begin
     process (clk)
     begin
         if (clk'event AND clk='1') then
            q<=d;
         end if;
      end process;
      qbar<=NOT q;
 end one_dff;
 ---------------------------------
```



生成一个寄存器

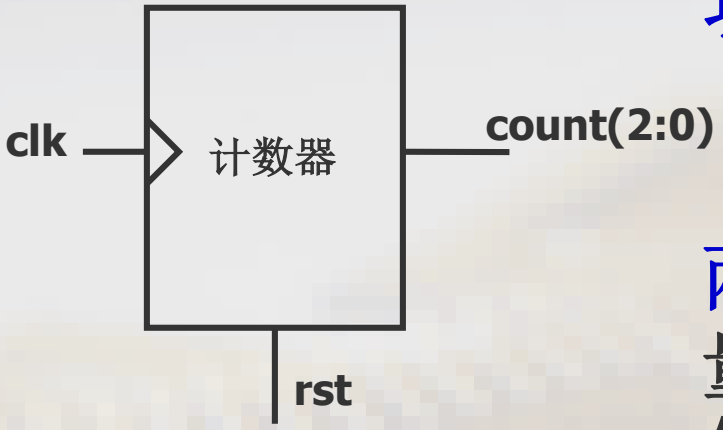
使用组合逻辑，而不是寄存器

例 7.6 小结

如果不对代码进行有效合理的组织，则会生成额外的寄存器。

另外，在一些FPGA/CPLD器件中，对同一个代码，综合工具与布局布线工具的报告文件显示寄存器的数目可能是不一致的。

例7.7 模8计数器



功能描述：

模8计数器。

两种实现方案：方案一使用变量进行同步赋值，方案二使用信号进行同步赋值。

方案1： 变量方式

不必声明包集

```
entity counter is
  port( clk, rst: IN BIT;
        count: OUT integer range 0 to 7 );
end counter;
-----------------------------------------
architecture counter of counter is
 begin
   process (clk, rst)
     variable temp: integer range 0 to 7;
   begin
     if (rst='1') then
      temp:=0;
     elsif (clk'event AND clk='1')  then
         temp:=temp+1;
      end if;
   count<=temp;
   end process;
end counter;
```

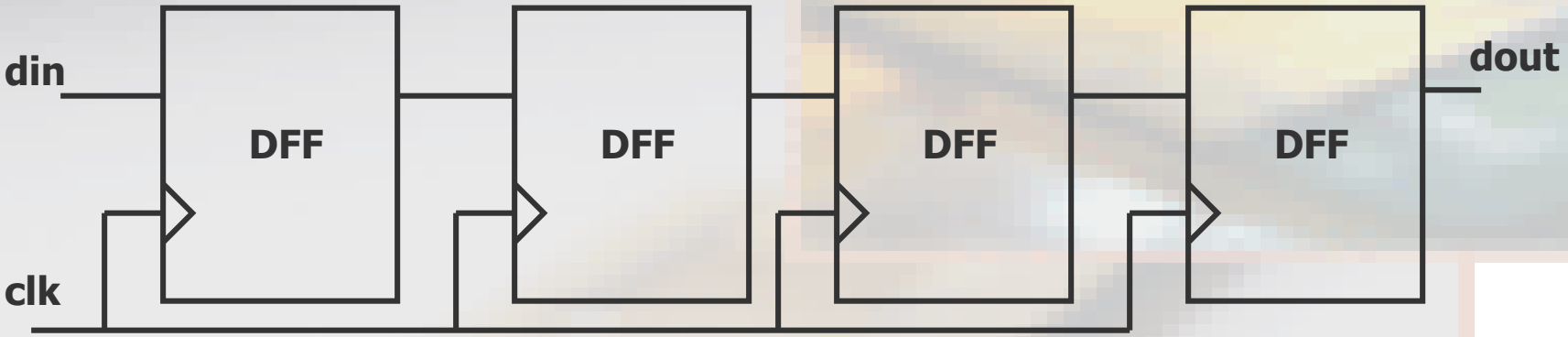
方案2： 信号方式

```
entity counter is
  port( rst, clk: IN BIT;
        count: BUFFER integer range 0 to 7 );
end counter;
-----------------------------------------
architecture counter of counter is
 begin
    process (clk, rst)
    begin
       if (rst='1') then
          count<=0;
       elsif (clk'event AND clk='1') then
          count<=count+1;
       end if;
    end process;
end counter;
```

buffer类型端口

都需要使用寄存器来保存3位的当前计数值，
综合时均会生成3个寄存器。

例7.8 移位寄存器



三个设计方案：

- 三个变量顺序排列
- 三个信号顺序排列
- 三个变量颠倒顺序排列

方案1： 三个变量顺序排列

```
entity shift is
  port( clk, din: IN BIT;
        dout: OUT BIT );
end shift;
-----------------------
architecture shi
begin
   process (clk)
     variable a, b, c:
   begin
      if (clk'event A    D clk='1')  then
         dout<=c;
         c:=b;
         b:=a;
         a:=din;
      end if;
   end process;
end shift;
```

不必声明包集

c、b、a的值被赋给dout、c、b之前没有被赋过值，综合时会生成3个寄存器，用以存储process上一次执行后c、b、a的值。

生成1个寄存器

方案2： 三个信号顺序排列

```
entity shift is
  port( din, clk: IN BIT;
        dout: OUT BIT );
end shift;
----------------------------------------
hitecture shift of shift is
 signal a, b, c: BIT;
gin
 process (clk)
 begin
    if (clk'event AND clk='1') then
       a<=din;
       b<=a;
       c<=b;
       dout<=c;
    end if;
 end process;
end shift;
```

生成4个寄存器

方案一、二的波形相同，dout较din延后4个时钟周期

方案3： 三个变量颠倒顺序排列

```
entity shift is
   port( clk, din: IN BIT;
         dout: OUT BIT );
end shift;
-----------------------------------
architecture shift of shift is
 begin
    process (clk)
      variable a, b, c: BIT;
    begin
       if (clk'event A
          a:=din;
          b:=a;
          c:=b;
          dout<=c;
       end if;
    end process;
 end shift;
```

赋值顺序与方案一相反，因对变量的赋值是立即生效的，三行代码可以简化为c:=din。

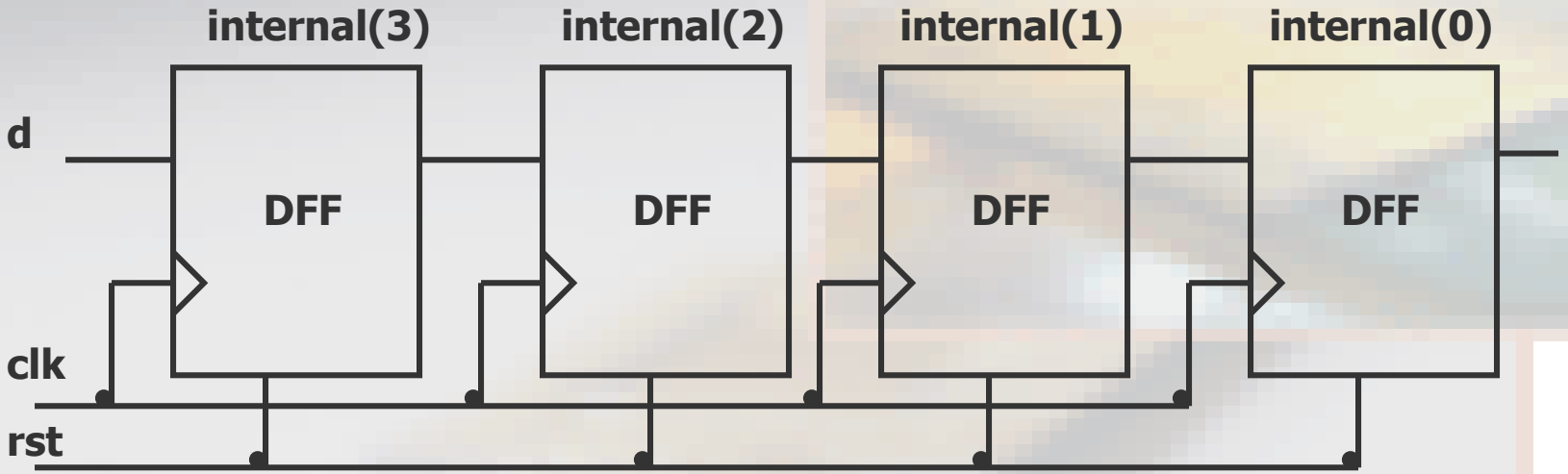
整个四行代码可以简化为 dout<=din;

方案1： 三个变量顺序排列

```
-----------------------------------------
entity shift is
   port( clk, din: IN BIT;
         dout: OUT BIT );
end shift;
-----------------------------------------
architecture shift of shift is
 begin
    process (clk)
      variable a, b, c: BIT;
    egin
      if (clk'event AND clk='1')  then
          dout<=c;
          c:=b;
          b:=a;
          a:=din;
       end if;
    end process;
 end shift;
```

方案三中dout较din延后1个时钟周期，未能实现设计初衷

例7.9 移位寄存器#2—增加了一个复位端



两种设计方案：

- 用信号生成寄存器
- 用变量生成寄存器

方案1： 内部信号方式

```
library ieee;
use ieee.std_logic_1164.all;
entity shiftreg is
   port( d, clk, rst: IN std_logic;
         q: OUT std_logic );
end shiftreg;
architecture behavior of shiftreg is
    signal internal: std_logic_vector(3
    downto 0);
begin
   process (clk, rst)
   begin
       if (rst='1') then
           internal<=(ohters=>'0');
       elsif (clk'event AND clk='1')  then
         internal<=d & internal(3 donwto 1);
       end if;
   end process;
   q<=internal(0);
end behavior;
```

方案2： 内部变量方式

```
library ieee;
use ieee.std_logic_1164.all;
entity shiftreg is
   port( d, clk, rst: IN std_logic;
         q: OUT std_logic );
end shiftreg;
architecture behavior of shiftreg is
    variable internal: std_logic_vector(3
    downto 0);
begin
   process (clk, rst)
   begin
       if (rst='1') then
           internal:=(ohters=>'0');
       elsif (clk'event AND clk='1')  then
         internal:=d & internal(3 donwto 1);
       end if;
   end process;
   q<=internal(0);
end behavior;
```

生成4个寄存器，输出比输入延后四个时钟周期

第7章　思考题

1、信号与变量的主要区别有哪些？

2、信号与变量是怎样影响寄存器数量的？

作业：7.1、7.3