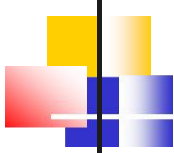
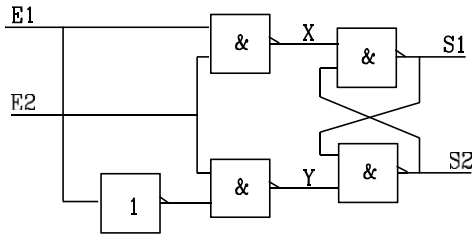


C4-顺序指令

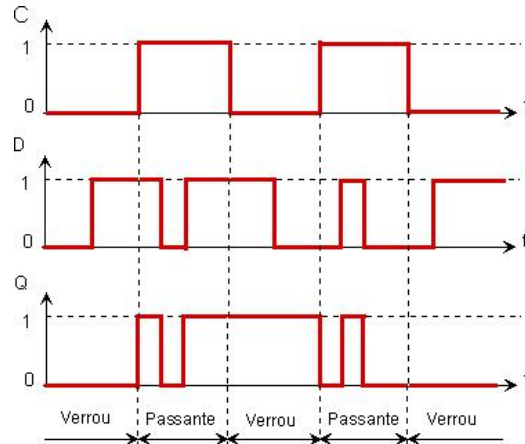
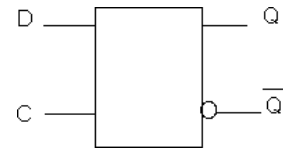
Yann Douze
VHDL

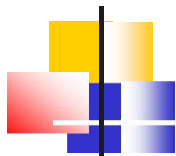


复习：D锁存器-电平触发

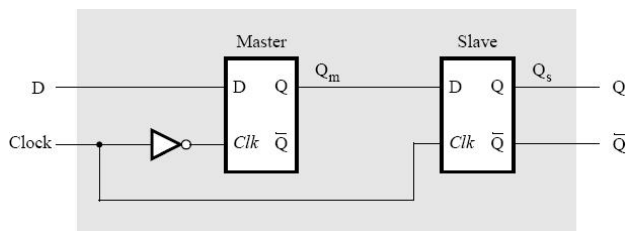


Entrées		Sorties	
C	D	Q_{n+1}	\overline{Q}_{n+1}
0	X	Q_n	\overline{Q}_n
1	0	0	1
1	1	1	0

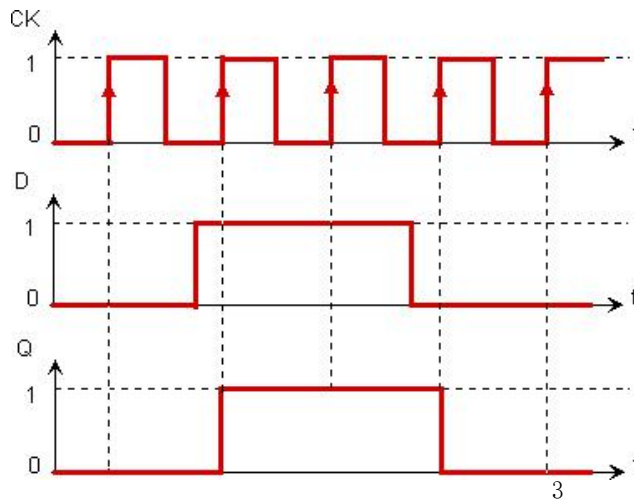
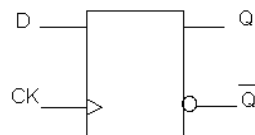




复习：D触发器-边沿出发



Entrées		Sorties	
CK	D	Q_{n+1}	\overline{Q}_{n+1}
0	X	Q_n	\overline{Q}_n
1	X	Q_n	\overline{Q}_n
↓	X	Q_n	\overline{Q}_n
↑	0	0	1
↑	1	1	0



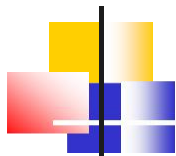


process 定义

- **process** 是电路描述的一部分，其中指令依次执行：一个接一个。
- 它允许使用结构化编程的标准指令对信号进行操作，就像在微处理器系统中一样。

语法：

```
[Nom_du_process :]process(Liste_de_sensibilité_nom_des_signaux)  
Begin  
    -- instructions du process  
end process [Nom_du_process] ;
```



Process运行规则

- 当敏感列表中信号的状态发生变化时，就会执行一个进程。
- 进程指令按顺序执行。
- 进程指令对信号状态的改变，在进程结束时才被执行。

在进程中，可以允许同一信号有多个驱动源(赋值源)，即在同一进程中存在多个同名的信号被赋值，其结果只有最后的赋值语句被启动，并进行赋值操作。



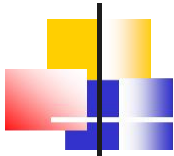
进程的作用

- 进程有三种不同的方法：
 1. 用结构化编程的指令描述组合电路: `if`, `case`等。
 2. 用于描述具有一个或多个存储单元（DFF触发器）的同步电路。
 3. 描述不可综合的功能，如 `testbench` 或建模。



顺序指令

- 注意！
- if和case语句只存在，并且只能存在于进程中。



IF指令

语法:

```
if condition then instructions  
[elsif condition then instructions]  
[else instructions]  
end if ;
```

Exemple:

```
Process (A,B,E1,E2,E3)  
Begin  
  if A='1' then SORTIE <= E1;  
  Elsif B = '1' then SORTIE <= E2;  
  Else SORTIE <= E3;  
  end if ;  
end process ;
```

注意:if语句只能在进程中使用



case 指令

语法:

CASE expression IS

WHEN choix => instructions;

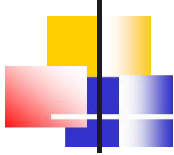
[WHEN choix => instructions;]

[WHEN OTHERS => instructions;]

END CASE;

除非所列条件的取值能完整覆盖case语句所有取值，否则other不应省略。
综合器会插入不必要的锁存器。

注意: 与if语句一样，CASE语句只能在进程中使用。



示例

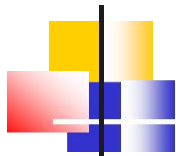
```
process (TEST,A,B)
begin
    case TEST is
        when "00" => F <= A and B;
        when "01" => F <= A or B;
        when "10" => F <= A xor B;
        when "11" => F <= A nand B;
        when others => F <= '0';
    end case;
end process;
```



组合进程

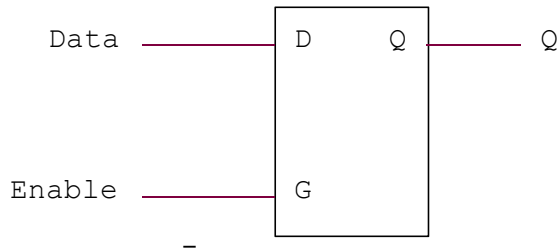
```
process (SEL, A, B, C)
begin
    if SEL = '1' then
        OP <= A and B;
    else
        OP <= C;
    end if;
end process;
```

- 敏感信号列表必须是完整的：进程中读取的所有信号都必须出现在敏感信号列表中。
- 在任何情况下都应输出赋值，以避免综合出D锁存器。



不完全条件赋值

```
process (Enable, Data)
begin
    if Enable = '1' then
        Q <= Data;
    end if;
end process;
```



- 不完全条件赋值生成的D锁存器（透明锁存器），这并不是所期待的设计。
- 有些FPGA没有D锁存器，因此会在组合逻辑上创建异步循环（这是要避免的！）



默认赋值**

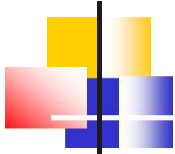
- 为了避免锁定透明片，建议使用默认赋值。

```
Process (SELA, SELB, A, B)
begin
  OP <= '0';
  if SELA = '1' then
    OP <= A;
  end if;
  if SELB = '1' then
    OP <= B;
  end if;
end process;
```

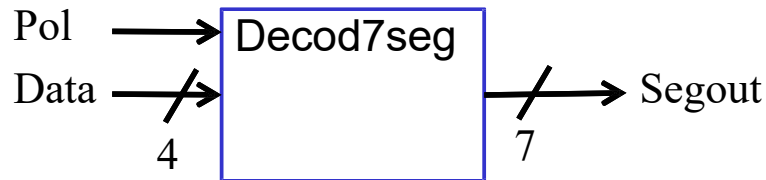
Assignement par défaut

Remplace l'événement par défaut

Remplace l'événement à nouveau

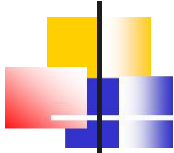


示例:7段译码器

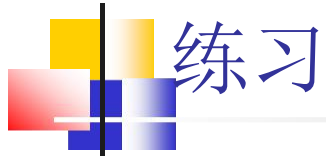


```
Entity SEVEN_SEG is
port(
  Data : in  std_logic_vector(3 downto 0); --Expected within 0...9
  Pol   : in  std_logic;                  -- '0' if active LOW
  Segout: out std_logic_vector(1 to 7)); --Segments A,B,C,D,E,F,G
end entity;
```

7段译码器组合架构

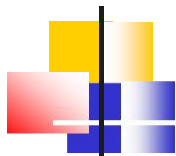


```
architecture COMB of SEVEN_SEG is
    signal sevseg : std_logic_vector(1 to 7);
begin
    Process(Data, Pol, sevseg)
    Begin
        case(Data) is
            when x"0"    => sevseg <= "1111110";
            when x"1"    => sevseg <= "0110000";
            when x"2"    => sevseg <= "1101101";
            when x"3"    => sevseg <= "1111001";
            when x"4"    => sevseg <= "0110011";
            when x"5"    => sevseg <= "1011011";
            when x"6"    => sevseg <= "1011111";
            when x"7"    => sevseg <= "1110000";
            when x"8"    => sevseg <= "1111111";
            when x"9"    => sevseg <= "1111011";
            when others => sevseg <= (others => '-');
        end case;
        if (Pol='1') then Segout
            <= sevseg;
        else
            Segout <= not(sevseg);
        end if;
    End process;
End architecture;
```



练习

- 练习1:8选1数据选择器



同步进程

```
process (CLK)
begin
  if RISING_EDGE (CLK) then
    Q1 <= D;
  end if;
end process;
```

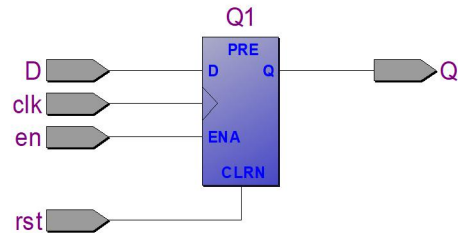
```
process (CLK)
begin
  if FALLING_EDGE (CLK) then
    Q2 <= D;
  end if;
end process;
```

- 同步进程是在时钟的每个上升沿执行。
- 要测试时钟的上升沿，请使用std_logic_1164包中定义的rising_edge()函数。
- 当时钟从状态‘0’变为状态‘1’时，RISING_EDGE为true。
- 用于描述D触发器(DFF)的翻转。

如何添加异步重置？

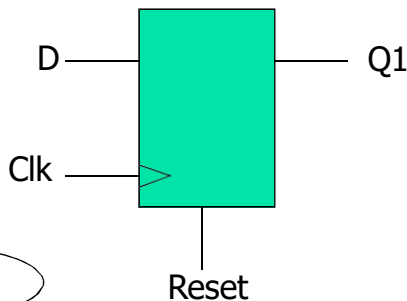


异步复位



```
process (Reset, Clk)
begin
  if Reset = '1' then
    Q1 <= '0';
  elsif RISING_EDGE(Clk) then
    Q1 <= D;
  end if;
end process;
```

•在时钟上升沿之前，检测异步复位。



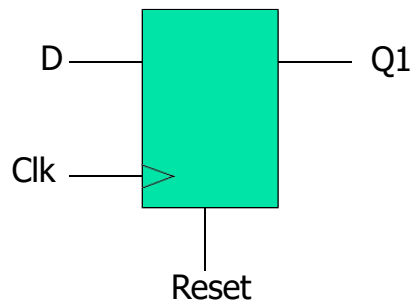
如何进行同步重置？

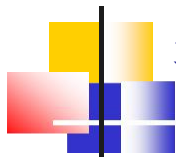


同步复位

```
process(Clk)
begin
  if RISING_EDGE(Clk) then
    if Reset = '1' then
      Q1 <= '0';
    else
      Q1 <= D;
    end if;
  end if;
end process;
```

- 在时钟上升沿之后，检测同步复位信号。





寄存器传输级 (RTL)

同步数字电路的
抽象模型

- 在同步进程中，每个赋值，都创建一个触发器。

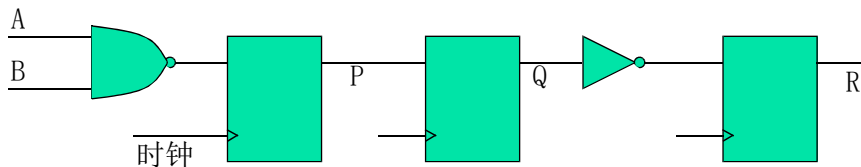
描述信号在硬件寄存器、存储器、组合逻辑与总线等逻辑单元之间流动。

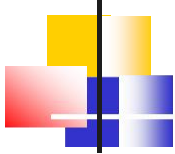
```
process (Clock)
begin
  if Rising_edge(Clock) then
    P <= A nand B;
    Q <= P;
    R <= not Q;
  end if;
end process;
```

Registers

Clock

Logique combinatoire

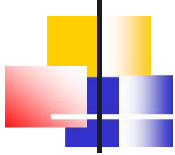




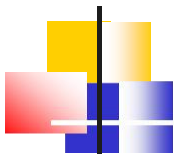
同步进程的规则

- 敏感信号列表应该包括时钟和复位信号，而不是其他的。
- 在进程中赋值的所有信号都必须通过复位信号来初始化。
- 要描述同步进程，请遵循以下结构：

```
process(clk,  
rst) begin  
  if rst = '1' then  
    -- valeur initiale  
  elsif rising_edge(clk) then  
    --  
    instruction  
  end if;  
end process;
```



- 注意，信号只有在进程结束时才被赋值。
- 完成4级移位寄存器的代码，并绘制时序图？



移位寄存器

```
-- Registre à décalage
entity registre_decalage is
port( Qin, rst, clk : in std_logic;
      Qout           : out std_logic);
end entity;
architecture RTL of registre_decalage is
    signal Q1,Q2,Q3 : std_logic;
begin
    process(Rst,Clk)
    begin
        if Rst = '1' then
            Qout <= '0'; Q1<='0';Q2<='0';Q3<='0';
        Elsif rising_edge(clk) then
            Qout <= Q3;
            Q3 <= Q2;
            Q2 <= Q1;
            Q1 <= Qin;
        End if;
    End process;
End architecture;
```

产生4个DFF。



错误的进程形式（1）？

```
process(Clock,Reset)
Begin
    if Rising_edge(Reset) then
        ...
    elsif Rising_edge(Clock) then
        ...
    end if;
end process;
```

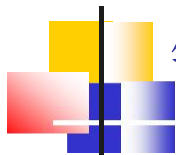
一个进程中引入了
两个边沿检测语句。

```
process(Clock,Reset)
begin
    if Reset = '1' then
        -- Actions asynchrone
    elsif Rising_edge(Clock) then
        -- Actions synchrone
    end if;
end process;
```

```
process(Clock,Reset,Ena)
begin
    if Reset = '1' then
        -- Actions asynchrone
    elsif Rising_edge(Clock) and Ena = '1' then
        -- Actions synchrone
    end if;
end process;
```

将用于产生寄存器的信号或变量的赋值语句放在了一个ELSE条件分支上。相当于检测，如果没有时钟信号时，则赋新值。显然不可能有这样的硬件电路与之对应。

```
process(All_Inputs)
begin
    -- Logique purement combinatoire
end process;
```

错误的进程形式（2）？

如果一个变量已在IF的边沿检测语句结构中作了赋值操作，就不能在同一进程中再作读操作。

一种错误是将边沿表达式当成了操作数。

IF NOT(clock'EVENT AND clock='1') THEN ...

```
process(Clock,Reset)
begin
    if Reset = '1' then
        -- Actions asynchrone
    elsif Rising_edge(Clock) then
        -- Actions synchrone
    end if;
    -- d'autres actions
end process;
```

```
process(Clock)
begin
    if Rising_edge(Clock) then
        -- Actions synchrone
    end if;
end process;
```

```
process(Clock,Reset)
begin
    if Reset = '0' then
        if Rising_edge(Clock) then
            -- Actions synchrone
        end if;
    else
        -- Actions asynchrone
    end if;
end process;
```

锁存器的引入是为了能使某种状态必须保持到下一次时钟沿到来。在边沿检测语句之外，要特别注意不要对已赋值的信号做读操作。否则，量不可能引出锁存器。不

错误将输出赋值信号放在了进程内部，将导致综合后的电路中多了两个不必要的寄存器。

PROCESS (CLK)

BEGIN

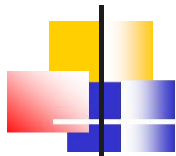
IF (CLK'EVENT AND CLK='1') THEN B <= C;

A <= B; H <= I;

I <= J XOR K;

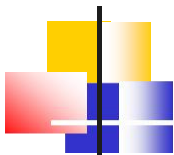
END IF;

END PROCESS ;



练习

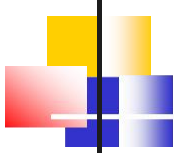
练习2： 串行OR或线性反馈移位寄存器LFSR



不可综合的进程

- 不可综合的进程用于：
 - 描述模型
 - 编写testbench
- 不可综合的进程没有敏感列表，这些是WAIT指令，可同步进程。

- 可以组合三种形式的**WAIT**
 - **WAIT ON** 事件； 示例： **wait on** A,B,C,D
 - 替换常见进程的敏感信号列表
 - **WAIT FOR** 时间； 示例： **wait for** 100 ns
 - 产生时间延迟
 - **WAIT UNTIL** 条件； 示例： **wait until** rst = '1'
 - 阻塞条件
 - **WAIT**;
 - 无限循环，类似C语言中的while (1)

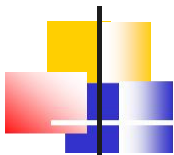


不可综合进程示例

```
-- Génération d'un reset
STIMULUS: process
begin
    reset <= '1';
    wait for 50 NS;
    reset <= '0';
    wait;
end process STIMULUS;
```

```
-- Génération d'une horloge
ClockGenerator: process
begin
    Clock <= '0';
    wait for 5 NS;
    Clock <= '1';
    wait for 5 NS;
end process;
```

两个进程同时使用的。



循环指令

```
for PARAMETER in LOOP_RANGE loop
    ...
end loop;
```

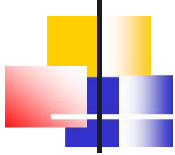
```
While CONDITION loop
    . 在进程内部。
    while 需要定义变量。
end loop;
```

```
boucle1: -- étiquette optionnelle
FOR i IN 0 TO 10 LOOP
    -- calcul des puissances de 2
    b := 2**i;
    WAIT FOR 10 ns; --toutes les 10 ns
END LOOP;
```

```
I := 0;
WHILE b < 1025 LOOP
    b := 2**i;
    i := i+1;
    WAIT FOR 10 ns;
END LOOP;
```

FOR后的循环变量是一个临时变量，属**LOOP**语句的局部变量，不必事先定义。这个变量只能作为赋值源，不能被赋值。它由**LOOP**语句自动定义，使用时应当注意在**LOOP**语句范围内不要再使用其它与此循环变量同名的标识符。

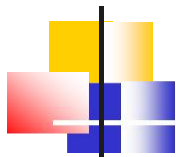
循环次数范围规定**LOOP**语句中的顺序语句被执行的次数。循环变量从循环次数范围的初值开始，每执行完一次顺序语句后递增1，直至达到循环次数范围指定的最大值。



循环示例

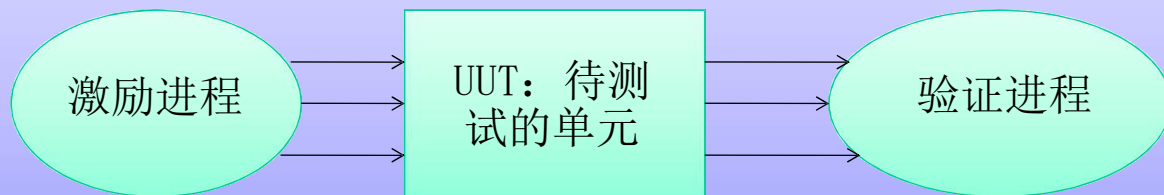
```
ClockGenerator: process  
begin  
    while not Stop loop  
        Clock <= '0';  
        wait for 5 NS;  
        Clock <= '1';  
        wait for 5 NS;  
    end loop;  
    wait;  
end process;
```

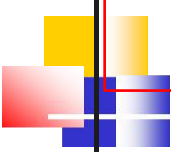
这里**stop**可能是自己定义的，



testbench

试验台

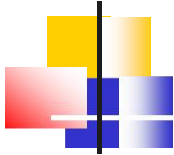




验证=断言

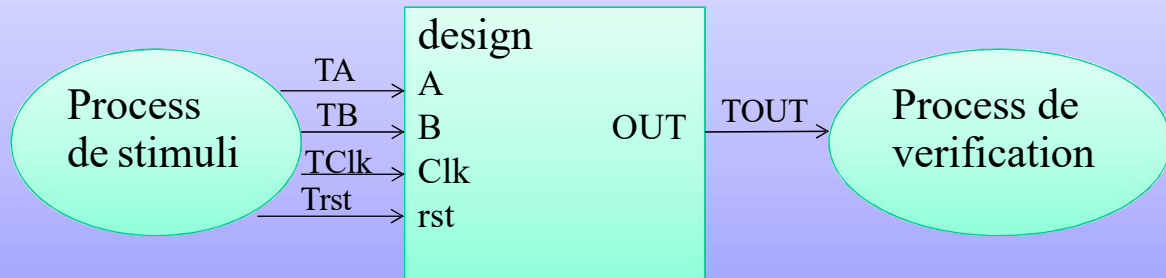
- `assert`: 允许在屏幕上写入消息。
- 语法:
 - `ASSERT condition REPORT message <SECURITY ...>`
- 强制消息:
 - `ASSERT FALSE REPORT "Toujours à l'écran " SEVERITY note;`
 - `REPORT "Toujours à l'écran " SEVERITY note;`
- 顺序内容测试:
 - `ASSERT s = '1' REPORT "la sortie vaut '0' et ce n'est pas normal " SEVERITY warning;`
--提出一个警告
 - `ASSERT s = '1' REPORT "la sortie vaut '0' et ce n'est pas normal " SEVERITY failure;`
--报告错误并停止执行

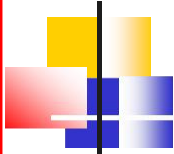
FAILURE 用在发生了致命错误，仿真过程必须立即停止的情况。



示例

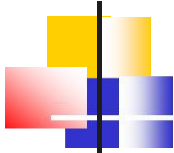
design_tb





testbench (1)

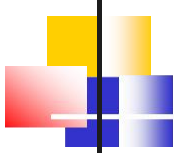
```
Entity design_tb is
End entity;
Architecture bench of design_tb is
    Signal Tclk : std_logic := '0';
    Signal Trst : std_logic;
    signal TA,TB,TOUT : std_logic_vector(3 downto 0);
    signal Done : boolean := False;
Begin
    -- instanciation du composant à tester
    UUT: entity work.design port map (
        A => TA, B => TB,
        OUT => TOUT,
        clk => Tclk, rst => Trst);
    -- Génération d'une horloge
    Tclk <= '0' when Done else not Tclk after 50 ns;
    -- Génération d'un reset au début
    Trst <= '1', '0' after 5 ns;
```



testbench (2)

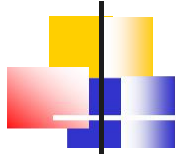
```
Stimuli: process
begin
    TA <= "0000";
    TB <= "0000";
    wait for 10 NS;
    TA <= "1111";
    wait for 10 NS;
    TB <= "1111";
    wait for 10 NS;
    TA <= "0101";
    TB <= "1010";
    wait;
end process;
```

```
Verification: process
begin
    wait for 5 NS;
    assert TOUT = "0000" report "erreur"
    severity warning;
    wait for 10 NS;
    assert TOUT = "1111" report "erreur"
    severity warning;
    wait for 10 NS;
    assert TOUT = "1111" report "erreur"
    severity warning;
    wait for 10 NS;
    assert TOUT = "1010" report "erreur"
    severity warning;
    Done <= True;
    wait;
end process;
End architecture;
```



Testbench: 备选方案

```
Stimuli_&_verification: process
Begin
TA <= "0000";
TB <= "0000";
wait for 5 NS;
assert TOUT = "0000" report "erreur" severity warning;
wait for 5 NS;
TA <= "1111";
wait for 5 NS;
assert TOUT = "1111" report "erreur" severity warning;
wait for 5 NS;
TA <= "0101";
TB <= "1010";
wait for 5 NS;
assert TOUT = "1010" report "erreur" severity warning;
Done <= True;
wait;
end process;
End architecture;
```



练习

- 练习3: MinMax电路

