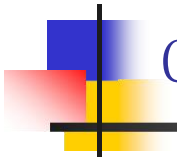
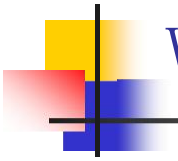


C2 VHDL描述的结构

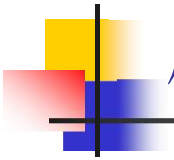
Yann Douze

VHDL



VHDL描述的结构

- VHDL描述由两个不可分割的部分组成，即：
  - 实体，它定义输入和输出。
  - 结构体，它包含允许执行预期操作的VHDL指令。
- 参见课程资料。



库声明

- 任何用于综合的VHDL描述都需要库。
- IEEE对它们进行了标准化，特别是IEEE 1164库。
- 它们包含信号类型的定义，元件、函数和子程序允许执行的算术和逻辑操作等。

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
```

实体声明

- *语法：*

```
entity NOM_DE_L_ENTITE is
    port ( Description des signaux d'entrées /sorties …);
end entity;
```

- Exemple :

```
entity SEQUENCEMENT is
    port (
          CLOCK : in std_logic;
          RESET : in std_logic;
          Q : out std_logic_vector(1 downto 0)
          );
end entity;
```


注意：在PORT语句的最后一个信号定义之后，不要使用分号。

I/O信号声明

- 端口语句:

Syntaxe: ***NOM_DU_SIGNAL :*** ***sens*** ***type;***

Exemple: ***CLOCK:*** ***in*** ***std_logic;***

BUS : ***out*** ***std_logic_vector (7 downto 0);***

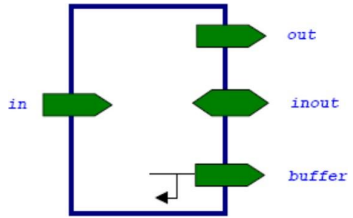
- 端口的方向:

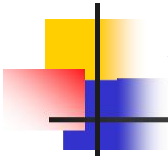
in：用于输入信号。

out：用于输出信号。

inout：用于输入输出信号

buffer：用于输出但可读取的信号（不推荐）。





类型

输入/输出信号类型为:

- 单个信号用***std_logic*** 。
- 由多个信号组成的总线用***std_logic_vector*** 。

例如，5位双向总线将表示如下:

LATCH : inout std_logic_vector (4 downto 0) ;

其中 ***LATCH(4)*** 对应于 **MSB** ， ***LATCH(0)*** 对应于 **LSB** 。

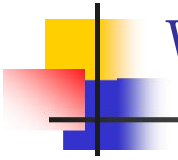
std_logic 类型的信号可以取的值为:

- ***"0"*** 或 ***"L"*** **:** 表示低电平。
- ***"1"***或 ***"H"*** **:** 表示高水平。
- ***"X"*** 或 ***"W"*** **:** 未知级别。
- ***'U'*** **:** 表示未初始化。
- ***"Z"*** **:** 高阻抗状态。
- ***"-"*** **:** 任意，即任意值。

结构体说明

- 结构体是相对于一个实体的。
- 它描述了设计的主体，它的行为，并通过指令建立了输入和输出之间的关系。
- 示例：

```
-- Opérateurs logiques de base
entity PORTES is
    port (A,B :in std_logic;
    Y1,Y2,Y3,Y4,Y5,Y6,Y7:out std_logic);
end entity;
architecture DESCRIPTION of PORTES is
begin
    Y1 <= A and B;
    Y2 <= A or B;
    Y3 <= A xor B;
    Y4 <= not A;
    Y5 <= A nand B;
    Y6 <= A nor B;
    Y7 <= not(A xor B);
end architecture;
```



VHDL：并行语言？

```
architecture DESCRIPTION of DECOD is
begin
     -- instructions concurrentes
    D0 <= (not(IN1) and not(IN0));        -- première instruction
    D1 <= (not(IN1) and IN0);             -- deuxième instruction
end architecture;
```

- 在结构体的开始和结束之间，处于一个相互并行的指令结构中。
- 并行指令：
  - 编写指令的顺序并不重要。。
  - 所有指令都被评估，并同时影响输出信号的时序。
  - 这是与计算机语言的主要区别。

下面的体系结构是等效的：

```
architecture DESCRIPTION of DECOD is
begin
    D1 <= (not(IN1) and IN0);             -- deuxième instruction
    D0 <= (not(IN1) AND not(IN0));        -- première instruction
end architecture;
```

VHDL描述示例：

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
```

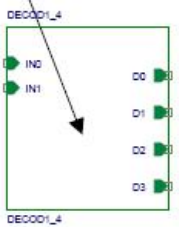
Déclaration des bibliothèques

Commentaires, en VHDL ils commencent par - -

```
-- décodeur
-- Un parmi quatre
```

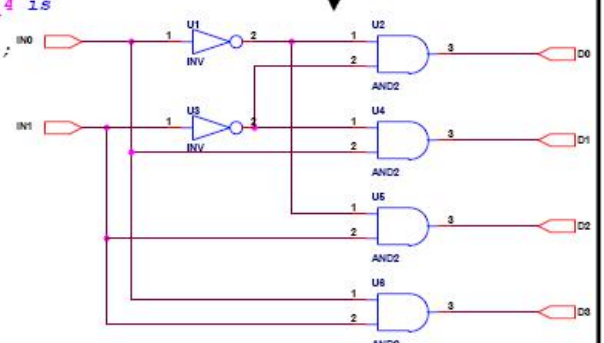
Déclaration de l'entité du décodeur
Correspondance schématique

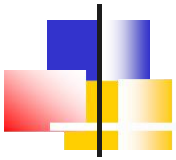
```
entity DECOD1_4 is
      port(IN0, IN1: in std_logic;
           D0, D1, D2, D3: out std_logic);
end DECOD1_4;
```



Déclaration de l'architecture du décodeur
Correspondance schématique

```
architecture DESCRIPTION of DECOD1_4 is
begin
      D0 <= (not(IN1) and not(IN0));
      D1 <= (not(IN1) and IN0);
      D2 <= (IN1 and not(IN0));
      D3 <= (IN1 and IN0);
end DESCRIPTION;
```

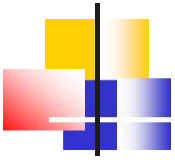




C3-基本运算符

Yann Douze

VHDL



简单赋值:<=

Exemples :

***S* <= *E2*;**

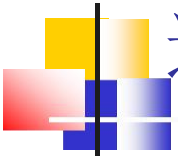
***S* <= *'0'*;**

***S* <= *'1'*;**

对于多位信号，使用省略号**"..."**，

BINARY，例如：BUS <= " 1001"; - BUS = 9 十进制

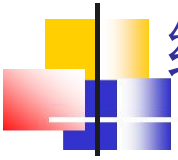
HEXA ，例如：BUS <= x" 9"; - BUS= 9 in 十六进制



逻辑运算符

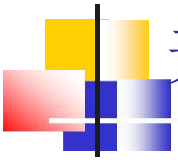
非	→	not
与	→	and
与非	→	nand
或	→	or
或非	→	nor
异或	→	xor

Exemple : S1 <= (E1 and E2) or (E3 nand E4);



练习

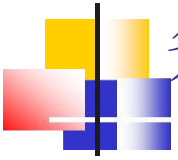
做练习1和2。



关系运算符

- 它们允许根据测试结果或条件改变信号的状态。

等于	€ =
不等于	€ /=
小于	€ <
小于等于	€ <=
大于	€ >
大于等于	€ >=

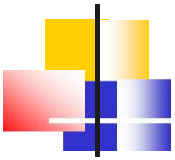


条件赋值语句

- 根据一个或多个信号、值、常量之间的逻辑条件的结果改变信号的赋值。

SIGNAL* <= *expression* *when* *condition
[else* *expression* *when* *condition]
[else* *expression];

注意：指令*[else expression]*允许定义
默认情况下为*SIGNAL*。



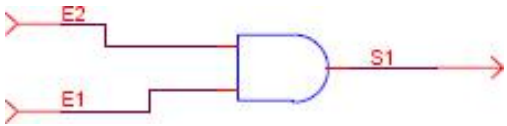
条件赋值（2）

例1:

--当（E1=‘1’）时，S1 取 E2 的值，否则 S1 取值‘0’，

```
S1 <= E2 when ( E1= '1') else '0';
```

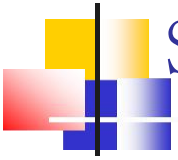
对应电路:与逻辑



例2:

--2选1数据选件器的行为描述

```
Y <= A when (SEL='0' ) else
     B when (SEL='1' ) else '0';
```



Selection赋值语句

该指令允许根据被选择信号的取值，为信号分配不同的取值。

```
with SIGNAL_DE_SELECTION select
    SIGNAL <= expression when valeur_de_selection,
     [expression when valeur_de_selection,]
     [expression when others];
```

注意:[Expression when others]语句不是必需的，但强烈建议使用，它允许设置为默认值。

Selection赋值语句 （2）

Exemple : *Multiplexeur 2 vers 1*

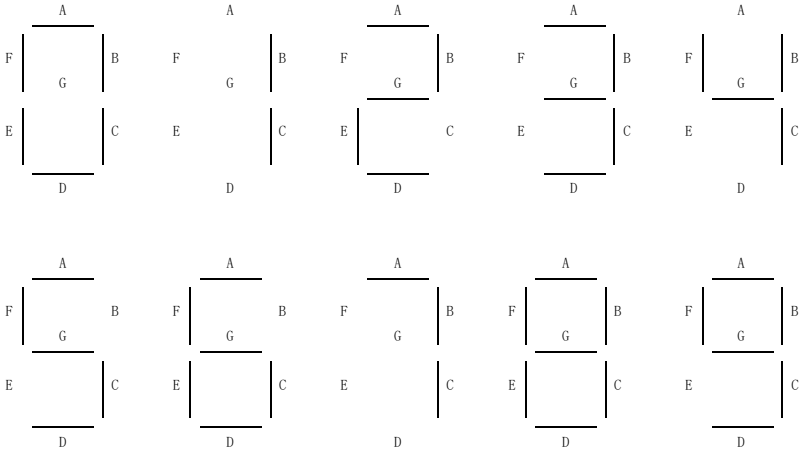
```
with SEL select
      Y <= A when '0',
      B when '1',
      '0' when others;
```

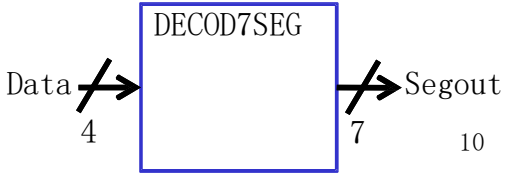
注意:在多路复用器的情况下， **when others**是必须的，因为对于所选择的信号其定义包含了其他的取值情况，所以应考虑其所有可能的值。

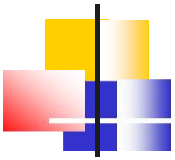
```
with SEL select
     Y <= A when '0',
      B when '1',
      '-' when others;
--对于 SEL 的其他情况，它将取任何值
--优化综合
```

示例:7段解码器（1）

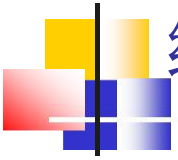
7段译码器-仅BCD（0..9）







示例:7段译码器（2）



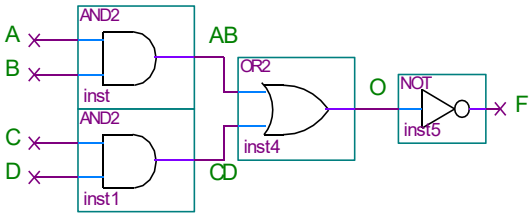
练习

- 练习3 使用选择性或条件赋值。

内部信号

- **Syntaxe :** **signal** ***NOM_DU_SIGNAL : type;***
- **Exemple :** **signal** ***I : std_logic;***
  signal BUS : std_logic_vector (7 downto 0);

下面的方案可以用两种不同的方式来描述:



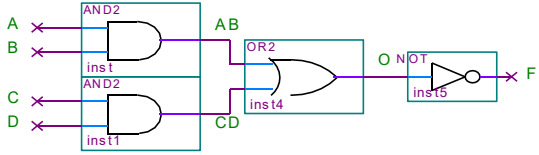
■无内部信号

```
architecture V1 of AOI is
Begin
   F <= not ((A and B) or (C and D));
end architecture V1;
```

内部信号（2），

- 具有内部信号

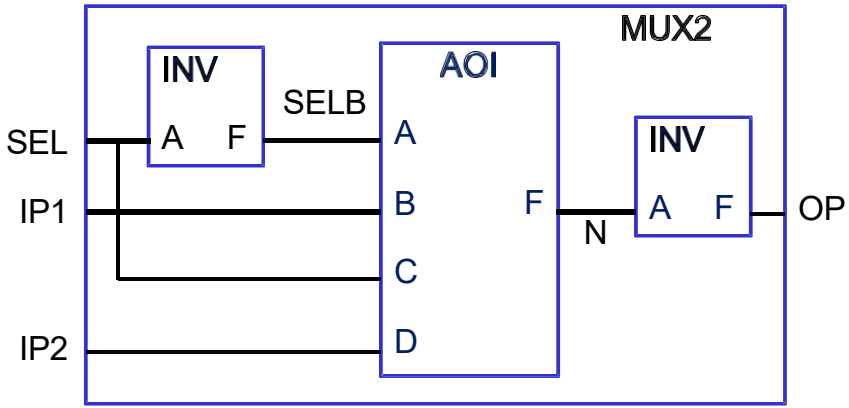
```
architecture V2 of AOI is
   --zone de declaration des signaux
   signal AB,CD,O: STD_LOGIC;
begin
   --instructions concurrentes
   AB <= A and B;
   CD <= C and D;
   O  <= AB or CD;
   F  <= not O;
end V2;
```

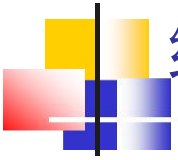


- 并发指令：
  - 写指示的顺序并不重要。
  - 所有指令都被评估并同时影响输出信号。
  - 与计算机语言的主要区别。

结构化描述

- 这是层次化连接的描述类型（NetList）
- 如果描述包含一个或多个元件，则该描述是结构性的。
- 示例:

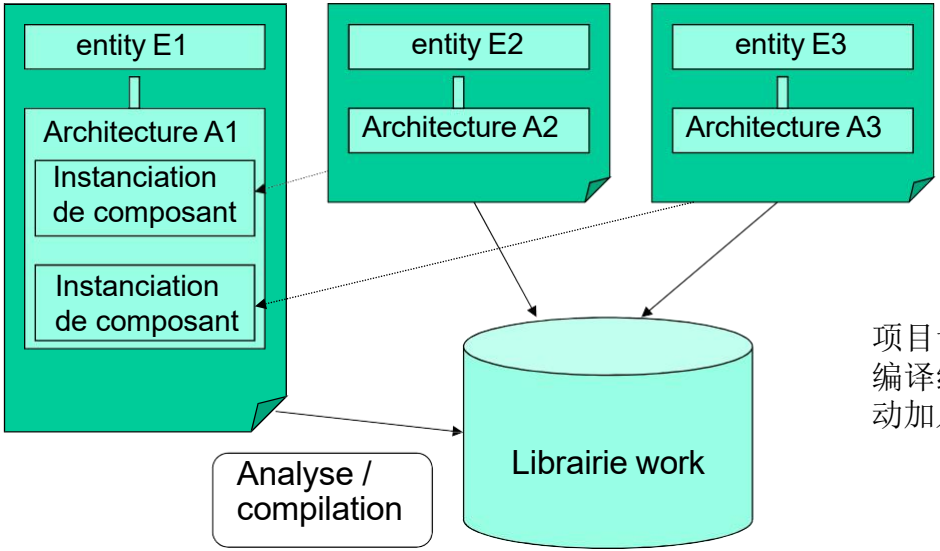




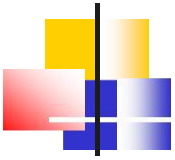
结构化描述

- **应遵循的步骤：**
  - 绘制要实例化的元件图。
  - 声明必要的内部信号列表：SIGNAL...
  - 实例化每个元件并指定其连接列表：PORT MAP...

元件例化

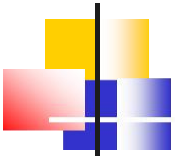


项目设计的元件经过编译综合后，会被自动加入到work库中



INV和AOI元件的声明

```
entity INV is
  port ( A   : in  STD_LOGIC;
         F   : out STD_LOGIC);
end entity;
architecture V1 of INV is
  ...
end architecture;

entity AOI is
  port ( A,B,C,D  : in  STD_LOGIC;
         F        : out STD_LOGIC);
end entity;
architecture V1 of AOI is
  ...
end architecture;
```

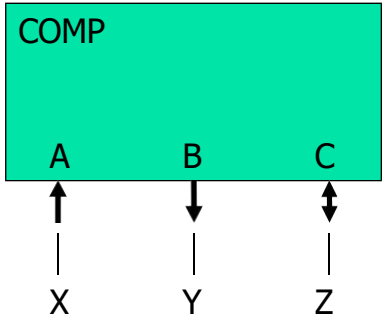


直接实例化

```
entity MUX2 is
   port ( SEL,IP1,IP2   : in  STD_LOGIC;
          op            : out STD_LOGIC);
end entity;
architecture DIRECTE of MUX2 is
   signal SELB, N: STD_LOGIC;
begin
   G1: entity WORK.INV(V1) port map (A => SEL, F => SELB);
   G2: entity WORK.AOI(V1) port map (
       A => SELB, B => IP1,
       C => SEL, D => IP2,
       F => N);
   G3: entity WORK.INV(V1) port map (A => N, F => OP);
end architecture;
```

通过名称或位置进行端口的关联

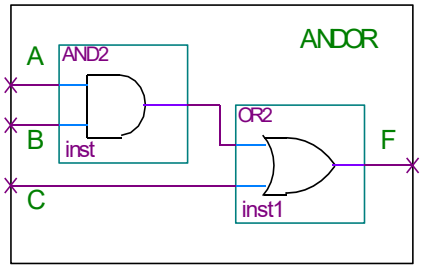
```
entity COMP
port(  A: in      STD_LOGIC;
       B: out     STD_LOGIC;
       C: inout   STD_LOGIC);
end entity;

Architecture V1 of COMP is
Signal X,Y,Z: STD_LOGIC;
begin
```




```
C1: entity work.COMP port map (A => X, B => Y, C => Z);
--association par nom
```

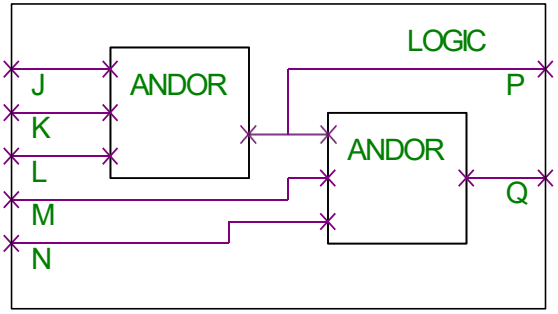
```
C1: entity work.COMP port map (X, Y, Z); --association par position
```

要完成的练习

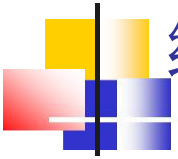



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ANDOR is
   port (
 A,B,C : in std_logic;
 F : out std_logic);
end entity;

Architecture dataflow of ANDOR is
Begin
   F <= (A and B) or C;
End architecture;
```

要完成的练习




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity LOGIC is
   port (
   J,K,L,M,N : in std_logic;
   P,Q : out std_logic);
end entity LOGIC;
Architecture STRUCT of LOGIC is
```



练习

- 做练习4和5。