N.N 1996

VHDL

Support de cours

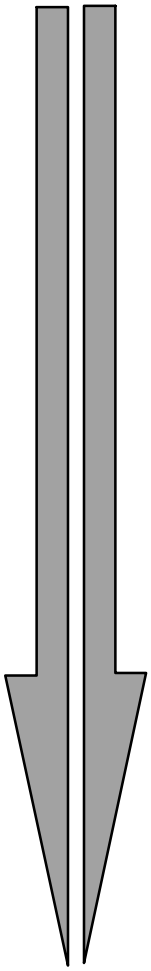
N.Nolhier
LAAS/CNRS
7, avenue du Colonel Roche
31077 TOULOUSE Cedex

Université Paul Sabatier 1997

• INTRODUCTION

VHDL

VHSIC
Hardware
Description
Language



Niveau d'abstraction

- **Composant élémentaire**
  (porte logique)
- **Circuits intégrés**
  (microprocesseur)
- **Cartes**
  (unité centrale)
- **Systèmes**
  (calculateur)
- **Processus**
  (système de navigation)

Pourquoi le VHDL ?

Demande des militaires US en 1980

- **nécessité d'un langage de description non ambigüe des systèmes matériels**
- **unicité d'une telle description**
- **pérennité du modèle**

En 1987 : VHDL devient une norme IEEE

Aujourd'hui c'est un standard du marché

- **outils de conception 100% VHDL (Synopsys)**
- **outils de conception plus ancien incluant une entrée VHDL (Cadence)**

Caractéristiques du VHDL

Avantages

- **Complexe**
- **Description structurée = travail en équipe**
- **Adapté aux projets multi-entreprises**
  modèles compilés -> sécurité
- **Indépendant de la technologie utilisée**

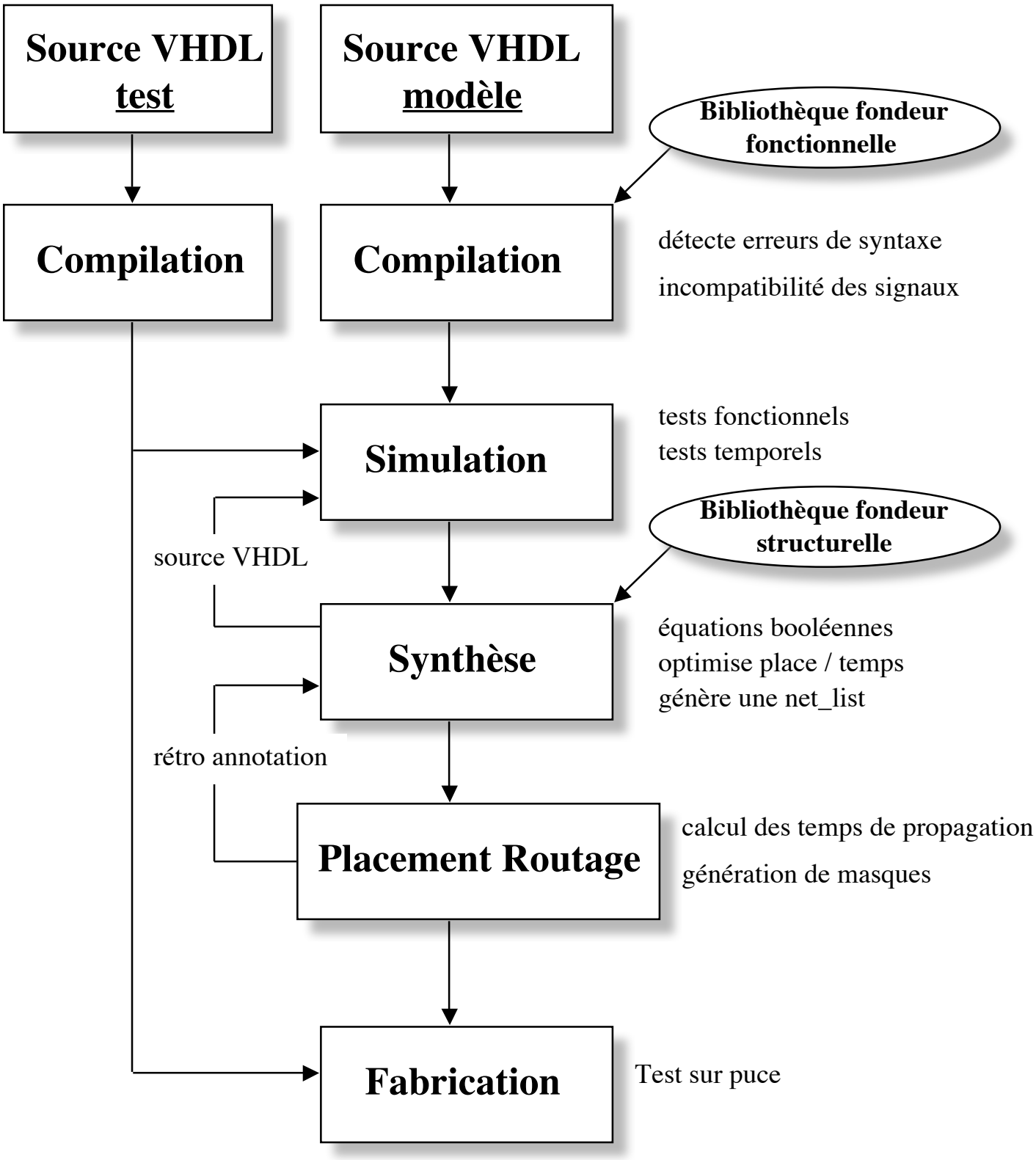
Inconvénients

- **Description complexe**
- **Tout n'est pas synthétisable**

Pentium : 3.1 Millions de transistors
1 an de conception

PowerPC 601 : 2.8 M

Conception d'un CI en VHDL



Structure d'un modèle VHDL

nom_du _modèle (signaux d'entrée, signaux de sortie)

architecture du modèle

- **définitions des signaux internes**
- **blocs de descriptions**

➠ **Plusieurs architectures peuvent être décrites**
choix à la compilation

➠ **Lien symbolique avec les éléments de bibliothèque**
redirection des connexions à la compilation

➠ **3 niveaux de descriptions peuvent être utilisés :**

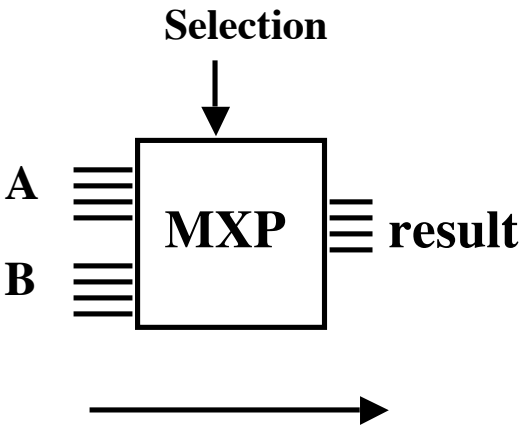
- ➥ **comportemental**
- ➥ **flot de données**
- ➥ **structurel**

<u>Ces niveaux sont concourants</u>

Les descriptions

Description comportementale

➠ **algorithmes séquentiels**




```
architecture nico of microP  is
.........
    signal A,B,result : bit_vector (1 to 4);
    selection : bit;
.........
MXP : process (Selection)
begin
    if (selection = '0') then
         result <= A after 10 ns;
    else
        result <= B after 15 ns;
    endif;
end process MXP;
........
```

VHDL autorise :

➥ **structures de boucle (loop endloop, while)**
➥ **branchements (if then else, case end case)**
➥ **appel aux procédures et fonctions**

Les descriptions (suite)

Description "flot de données"

➠ **utilisation d'équations concourantes**

```
architecture nico2 of microP  is
.........
    signal A,B,result : bit_vector (1 to 4);
    selection : bit;
.........
result <= A after 10 ns when selection = '0' else
              B after 15 ns;
```

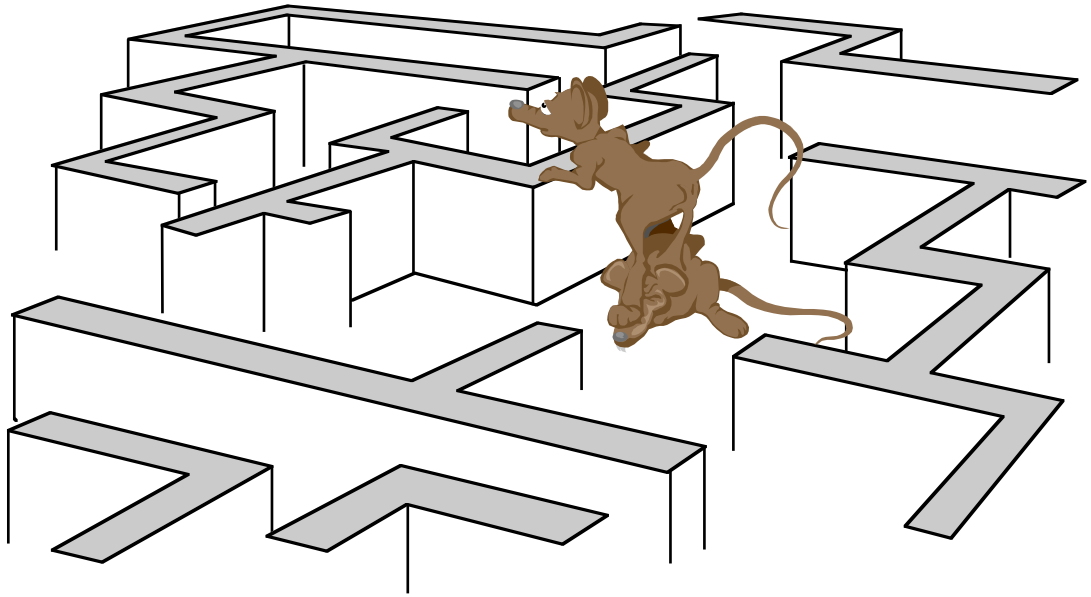
Description structurelle

➠ **création d'un lien avec un autre modèle**

- **ce modèle peut être**
  - **soit un élément de la bibliothèque fondeur**
  - **soit un autre modèle de l'utilisateur**

```
architecture nico3 of microP  is
.........
    signal A,B,result : bit_vector (1 to 4);
    selection : bit;
.........
G1 : MULTI4 port map ( A,B, selection, result);
```

• LA SEMANTIQUE DU LANGAGE



Les mots réservés

abs	**component**	**generic**	**next**	**record**	**use**
access	**configuration**	**guarded**	**nor**	**register**	
after	**constant**		**not**	**rem**	**variable**
alias		**if**	**null**	**report**	
all	**disconnect**	**in**		**return**	**wait**
and	**downto**	**inout**	**of**		**when**
architecture		**is**	**on**	**select**	**while**
array	**else**		**open**	**severity**	**with**
assert	**elsif**	**label**	**or**	**signal**	
attribute	**end**	**library**	**others**	**subtype**	**xor**
	entity	**linkage**	**out**		
begin	**exit**	**loop**		**then**	
block			**package**	**to**	
body	**file**	**map**	**port**	**transport**	
buffer	**for**	**mod**	**procedure**	**type**	
bus	**function**		**process**		
		nand		**units**	
case	**generate**	**new**	**range**	**until**	

Les identifieurs

- **Suite de lettres, chiffres et underscore "_"**
- **Le premier caractère doit être une lettre**
- **Tous les caractères sont significatifs**
- **VHDL ne fait aucune différence entre majuscules et minuscules**

Corrects : NON_ET Bascule_JK NE555

Interdits : A#2 2A A$2 A__2

Les commentaires

- **Ils sont liés à la ligne**
- **Ils commencent par deux tirets "--" et finissent à la fin de la ligne**
- **Ils sont nécessaires à la compréhension du modèle mais totalement ignorés par le compilateur**

```
-- Ceci est un commentaire !!!
q <= data2 after 10 ns; -- affectation de la sortie

-- attention l'instruction suivante sera ignorée :  q <= data2 after 10 ns;
```

Littéraux

- **Valeurs décimales**

12	**0**	**1E6**	**Entiers**
12.0	**0.0**	**0.456**	**Réels**
1.23E-12		**1.0e+6**	**Réels avec exposant**

- **Notation basée**

format : base#valeur#
base comprise entre 2 et 16

2#11111111# **16#FF#** **Entiers de valeur 255**

- **Valeurs physiques**

100 ps **3 ns** **5 v**
Il faut toujours un espace entre la valeur et l'unité

Littéraux (suite)

- **Caractères**

  Notés entre apostrophes

  'x' 'P' '2' ''' ' '

- **Chaines de caractères**

  Notés entre guillemets, minuscules et majuscules y sont significatives

  "Bonjour..." "" "23BVC$$Ld"
  "Il est possible de noter"& -- concaténation
  "sur 2 lignes"

- **Chaines de bits**

  Utilisés pour affecter des signaux de type bit_vector
  X"FFF" -- base hexadécimale, longeur 12 bits
  B"1111_1111_1111" -- base binaire , longeur 12 bits
  O"17" -- base octale, longeur 6 bits

Les objets et leurs types

- **VHDL permet de manipuler des objets typés**
- **un objet est le contenant d'une valeur d'un type donné**
- **4 classes d'objets :**
  - **CONSTANT : objet possédant une valeur fixe**
  - **VARIABLE : peut évoluer pendant la simulation**
  - **SIGNAL : variable + notion temporelle (valeurs datées)**
  - **FILE : ensemble de valeurs qui peuvent être lues ou écrites**
- **4 sortes de types :**
  - **Scalaires : entiers, réels...**
  - **Composites : tableaux, articles**
  - **Pointeurs**
  - **Fichiers**

Les types sont prédéfinis (package STANDARD) ou personnels
Ils sont statiques durant la simulation

Les types scalaires

- **Types entiers**

```
type integer is range  -2_147_483_648 to 2_147_483_647;  -- machine 32 bits
type Index is range ( 0 to 15);
```

- **Types flottants**

```
type real is range  -lim_inf to +lim_sup;                    -- prédéfini
type mon_real is range ( 0.0 to 13.8);
```

- **Types énumérés**

```
type bit is ('0','1');                                       -- prédéfini
type boolean is (false,true);                                -- prédéfini
type feu_tricolore is (vert,orange,rouge,panne);
type LOGIC4 is ('X','0','1','Z');
```

Les types scalaires (suite)

- **Types physiques**

```
Type volume is range integer'low to integer'high
    Units
        ml;
        cl      =       10 ml;
        dl      =       10 cl;
        l       =       10 dl;
        gallon  =       4546 ml;
    End units;
```

```
constant  PI : real := 3.141592;

variable  A,B,C : Integer;
variable  DATA : integer := 0;
variable  grand_rond : feu_tricolore := rouge;

signal    enable : Logic4 := 'Z';
signal    feu1,feu2 : feu_tricolore := panne;
```

Quand les objets ne sont pas initialisés, ils prennent la valeur la plus basse du type

Les types tableaux (array)

- **Le type Array réunit des objets de même type**
- **Un tableau se caractérise par :**
  - **sa dimension**
  - **le type de l'indice dans chacune de ses dimensions**
  - **le type de ses éléments**
- **Chaque élément est accessible par sa position (indice)**
- **on peut définir un type de tableaux non contraints, mais la dimension doit être spécifiée lors de la déclaration d'un objet de ce type**

```
Type string is array (Positive range <>) of character;      -- prédéfini
Type bit_vector is array (Natural range <>) of bit;         -- prédéfini

Type Compte_feu is array (Feu_tricolore) of integer;
Type Matrice2D is array (Positive range <>,Positive range <>) of
     real;
Signal mat3*4 : Matrice2D ( 1 to 3, 1 to 4);
Signal mot16 : bit_vector (15 downto 0);
```

Les types articles (Record)

- **Le type RECORD réunis des objets de types différents**
- **Les éléments(champs) sont accessibles par un nom**
- **La déclaration d'un article inclut la déclaration de chacun de ses champs**

```
Type type_mois is (jan,fev,mars,avr,mai,juin,juil,aout,sept,oct,nov,dec);

Type date is record
                mois : type_mois;
                annee : natural;
End record;

Type personne is record
                nom : string;
                prenom : string;
                age : natural;
                arrivee : date;
End record;
```

Les sous-types

- **Association d'une contrainte à un type**
- **La contrainte est optionnelle**
- **Le simulateur vérifie dynamiquement la valeur de l'objet**
- **Les opérateurs définis pour le type sont utilisables pour le sous-type**

```
Subtype printemps is type_mois range mars to juin ;
Subtype valeur is bit;
Subtype octect is  bit_vector ( 7 downto 0);
Subtype byte is bit_vector (7 downto 0);
```

Types prédéfinis

- **Déclarés dans le package STANDARD**

```
Type bit  is ('0','1');
Type boolean is (False,True);
Type character is (Null,....,Del);
Type Severity_Level is (Note,Warning,Error,Failure);

Type integer is ......(dépend de la machine);
Type real is ......(dépend de la machine);

Subtype natural is integer range 0 to integer'high;
Subtype positive is integer range 1 to integer'high;

Type string is array (positive range <> ) of character;
Type bit_vector is array (natural range <> ) of bit;
```

```
Type TIME is range
-9223372036854775808 to
  9223372036854775807
units fs;
        ps=1000 fs;
        ns=1000 ps;
        us=1000 ns;
        ms=1000 us;
        sec=1000 ms;
        min=60 sec;
        hr= 60 min;
end units;
```

Les attributs

- **associés à des types ou à des objets**
- **valeurs dynamiques**
- **prédéfinis mais possibilité d'attributs utilisateurs**
- **ils portent :**
  - **sur les types scalaires**
  - **sur les tableaux**
  - **sur des signaux**
- **Simplification de l'écriture et généricité**
- **Les attributs se note avec une ' après le type ou l'objet**
  - **ex : INTEGER'HIGH**

Les attributs sur types

```
type GAMME is (DO,RE,MI,FA,SOL,LA,SI);

subtype INDEX is integer range 16 downto 0 ;
```

INDEX'left	=16	GAMME'val(3)	=FA
INDEX'right	=0	GAMME'pos(FA)	=3
INDEX'low	=0	GAMME'pos(DO)	=0
INDEX'high	=16	GAMME'pred(RE)	=DO
GAMME'right	=SI	INDEX'succ(13)	=14
GAMME'high	=SI	INDEX'rightof(13)	=12
		GAMME'val(INDEX'leftof(0))	=RE

Le dépassement de dimension sur le type provoque une erreur :

GAMME'succ(SI) = erreur !!!

➥ pas de modulo

Les attributs sur tableaux

```
subtype OCTET is bit_vector (7 downto 0) ;
type MEMOIRE is array (positive  range <>, positive range <>) of bit ;

variable DATA : OCTET ;
variable ECRAN : MEMOIRE (1 to 640 , 1 to 480) ;
```

OCTET'left	= 7	ECRAN'right(1)	= 640
OCTET'high	= 7	ECRAN'right(2)	= 480
OCTET'range	: 7 downto 0	ECRAN'range(1)	: 1 to 640
OCTET'reverse_range	: 0 to 7	ECRAN'reverse_range(2)	: 480 dowto 1
OCTET'lenght	= 8	ECRAN'range(2)	: 1 to 480
DATA'right	= 0		

Les attributs ne s'appliquent pas sur un type tableau non contraint : MEMOIRE

Les opérateurs logiques

- **AND, NAND, OR, NOR, XOR et NOT**
- **utilisés pour des objets de type bit, booléen ou tableaux unidimenssionnels de ces types**
- **utilisation des opérateurs optimisée par le simulateur**
  - **ex: A and (B or C) et A=0 !!**

```
signal A,B,C,S       : bit;
variable raz, init, marche : boolean;

S <= A or ( B nand C ) after 10 ns;

raz := init and not marche ;
```

Les opérateurs relationnels

- **=, /=, >, <, >=, <=**
- **les opérateurs égalité et différent :définis sur tous les types (sauf file)**
- **inégalités s'appliquent à tous les types scalaires et les vecteurs d'élément entiers ou énumérés**
- **relation d'ordre sur les types énumérés : gauche->droite**

```
note1 := DO;
note2 := SI;

note1 < note2                -- true
"01001" < "100"              -- true comparaison caract/caract
B"01001" < B"100"            -- false
"VHDL" < "VHDL_"             -- true
"VHDL" /= "vhdl"             -- true
```

Les opérateurs arithmétiques

- **+, -, /, MOD, REM, ABS, ****
- **addition et soustraction définies pour tous les types numériques**
- **multiplication et division s'appliquent à 2 réels ou entiers ou 1 objet physique et un réeł entier**
- **MOD et REM définis pour le type entier**
- **** élève entier ou réel à une puissance entière**

```
signal     A,B,C: real;
signal     I,J,K : integer;

A <= B + C;                                   I <= J + K;
A <= B + real(J);                             I <= 5;   J <=- 2;
K <= I/J;              -- K = -2
K <= I MOD J;          -- K = -1 . K reçoit le signe de J
K <= I REM J;          -- K = 1 . K reçoit le signe de I
```

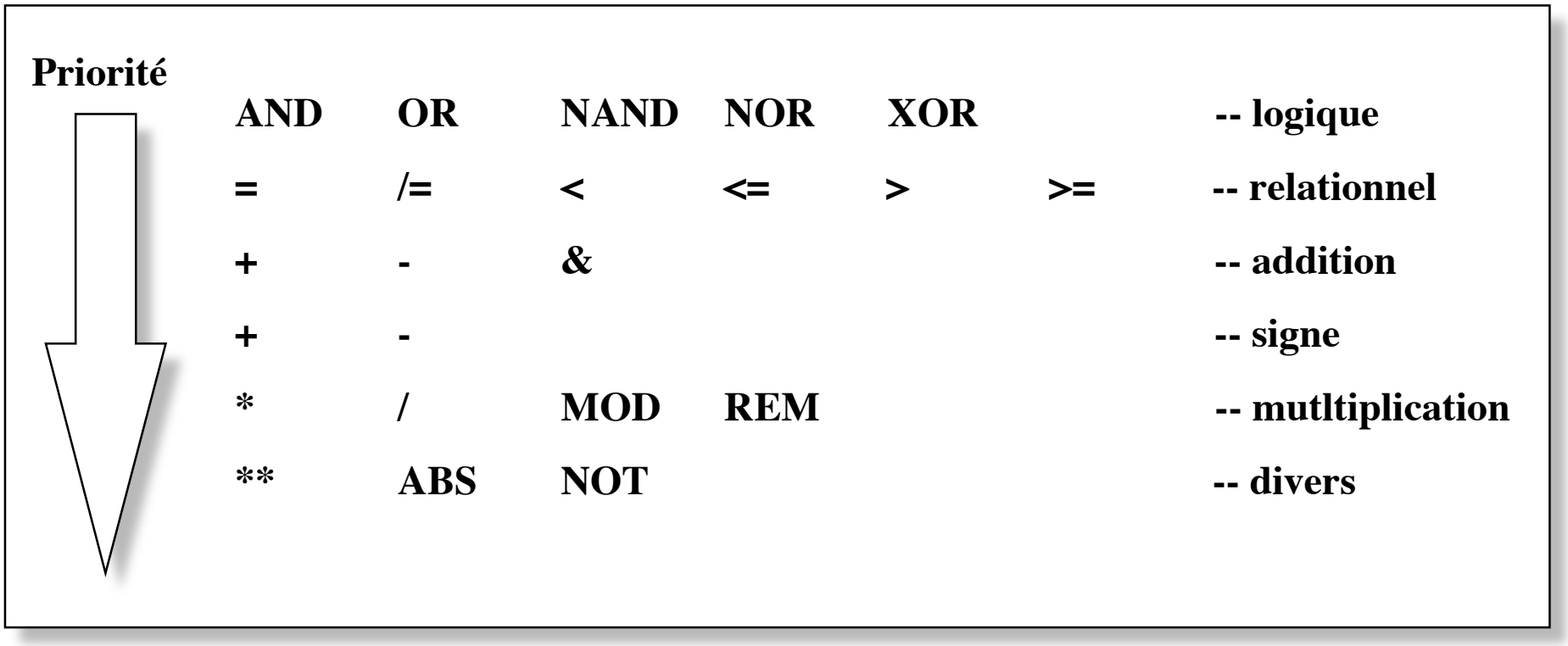
La concaténation

- **Utilise l'opérateur & et s'applique aux vecteurs de taille quelconque**
- **concaténation de 2 vecteurs, d'un élément et d'un vecteur ou de 2 éléments**

```
"CONCA"&"TENATION"          ->      "CONCATENATION"
'A' & 'B'                   ->      "AB"
'0' & "1011" & '0'          ->      "010110"

REG ( 31 downto 0 )         <=      GAUCHE & REG (31 downto 1);
```

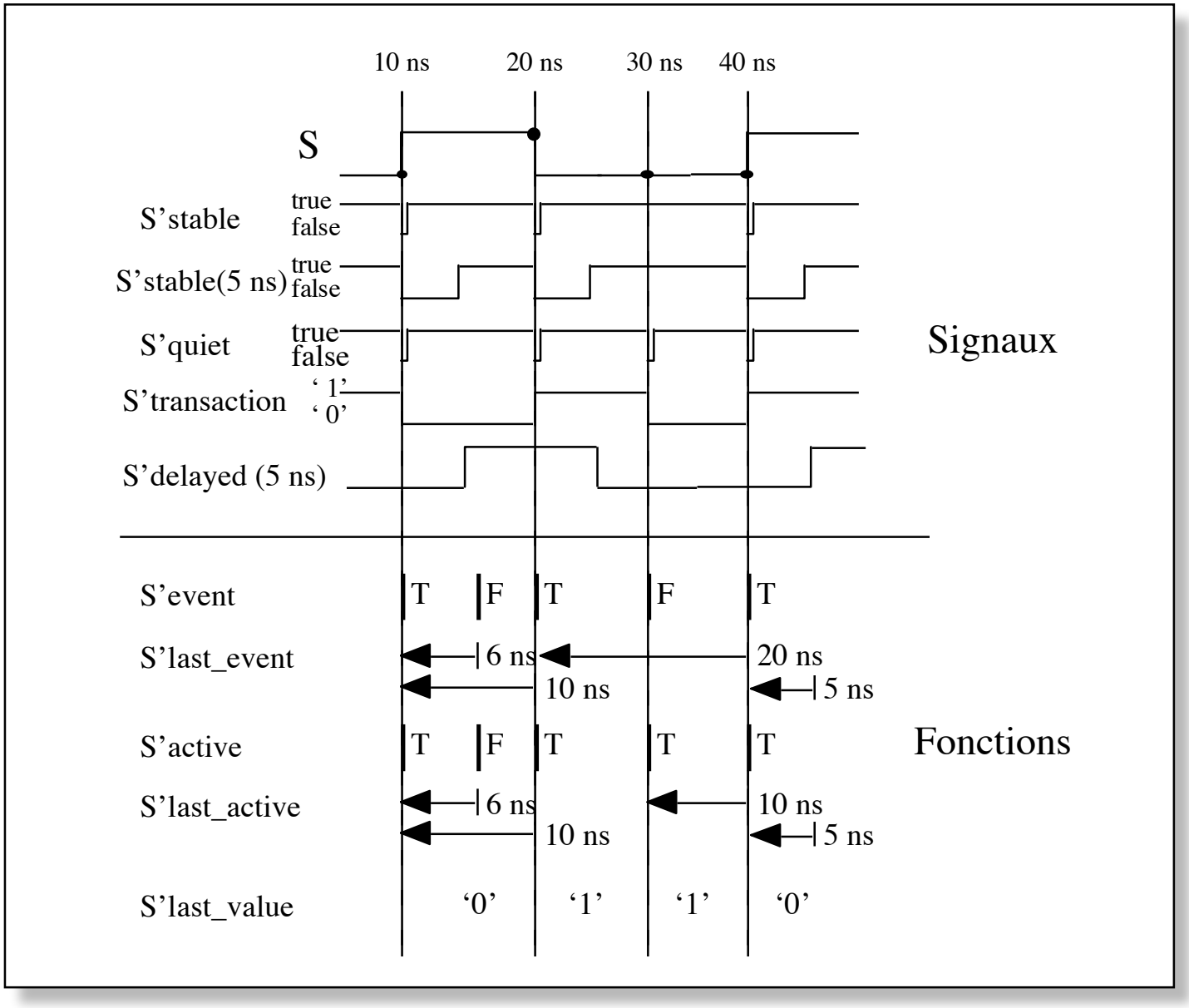
Priorité des opérateurs

- **6 classes de priorité :**



☞ **Utilisez les parenthèses !**
Possibilité de "surcharge" des opérateurs

Les attributs sur signaux



active ⇨ transaction

event ⇨ transition

Les agrégats

- **notation permettant de spécifier la valeur d'objets de type tableau ou article**
- **dans un agrégat les éléments sont spécifiés par association positionelle, nommée ou mixte**

```
variable client : personne ;
signal bus4 : Bit_vector ( 3 downto 0 );

client := ("Dupont","Michel", 45, ( janv,1992));          -- positionnelle
bus4 <= ('1','0','1','1');

bus4 <= (0=>'1', 3=>'1', 2=>'0', 1=>'1');                 -- nommée
signal reg : bit_vector (31 downto 0) := (5 => '1', 8 to 15 => '1', others => '0');
signal RAM : memoire (1 to 2048, 1 to 8) := (1 to 2048 => ( 1 to 8 => '1' ));

reg <= ('1','0','1', 15 => '1', others =>'0');            -- mixte
```

Instructions concurrentes

- **Le VHDL est un langage concurrent ➠ le système à modéliser peut être divisé en plusieurs blocs agissant en parallèle**
- **L'instruction concurrente est exécutée quand un évènement sur certains de ses signaux apparait**

  - ➠ **Instanciation de composants**
  - ➠ **Process**
  - ➠ **Affectation de signaux**
  - ➠ **Appel de procédure**
  - ➠ **Assertion**
  - ➠ **Bloc**
  - ➠ **Génération**

Instanciation de composants

- **Création d'un composant dont le modèle est défini ailleurs (hiérarchie)**

LABEL : NOM_DU_COMPOSANT port map (liste de signaux);

```
architecture structurelle of adder is
.....
begin
        C1 : PORTE_ET          port map (A1,A2,S1);
        C2 : PORTE_OU          port map (A1,S1,S2);
        C3 : INVERSEUR         port map (S2,S3);
        ...
......
end structurelle;
```

Process

- **Ne contient que des instructions séquentielles**
- **Un process "vit" toujours ➠ cyclique**
- **Doit être controlé par sa liste de sensibilité ou des synchronisations internes (wait)**

  ➥sinon bouclage

```
{LABEL :} process ( liste de sensibilité)
        déclarations
begin
        instructions séquentielles
        ........
end process {LABEL};
```

OU

```
{LABEL :}  process
        déclarations
begin
        wait on (liste de signaux)
        instructions séquentielles
        ........
end process {LABEL};
```

✓ **Ne pas mettre une liste de sensibilité et un (ou des) wait !!**

Les affectations

- **Affecter des valeurs à des signaux**

signal <= forme d'onde { délai }

```
C1 <= '1' after 10 ns;
C2 <= C1 after 1 ns, '0' after 20 ns;
C3 <= '1', '0' after 50 ns;
horloge <= not horloge after 20 ns;
```

Les affectations (suite)

•Affectation conditionnelle

{label:} SIGNAL <= forme d'onde 1 when condition 1 else
forme d'onde 2 when condition 2 else
.....
forme d'onde n ;

```
Sout <=  in1 after 5 ns when (selection = '1') else
         in2 after 6 ns;
```

•Affectation sélective

{label:} with expression select
SIGNAL <= forme d'onde 1 when choix 1,
forme d'onde 2 when choix 2,
.....
forme d'onde n when choix n ;

✔ **par défaut : when others**

```
type OPERATION is (ADD,SUBX,SUBY);
signal code_operatoire : operation;
....
with code_operatoire select
regA <=  X + Y when ADD,
         X - Y when SUBX,
         Y - X when SUBY;
```

L'appel de procédures

- **dans le cas d'un appel concurrent les paramètres sont des signaux ou des constantes**
- **sensible à un événement sur les paramètres d'entrée**

{label:} nom_de_la_procedure (liste des paramètres);

```
check_setup  (data, clock, setup_time);
check_hold  (data, clock, hold_time);
check_tw (clock, twl, twh);
```

Les assertions

- **Surveille une condition et génère un message si la condition est fausse**
- **on y associe un niveau d'erreur qui peut être pris en compte par le simulateur**

{label:} assert condition { report message } { severity niveau_d'erreur };

```
assert     S /= 'X'
report     " Le signal S est non_défini "
severity   warning ;

assert     ( frequence(clock) <= freq_max )
report     " clock est trop rapide "
severity   error ;
```

Les blocs

- **Similaires à un process mais contiennent des instructions concurrentes**
- **Le "block" permet :**
  - **la hiérarchisation d'intructions concurrentes et des déclarations locales**
  - **de synchroniser les instructions du bloc avec l'extérieur de ce dernier**

```
label : block { condition de garde }
        -- déclarations locales
        begin
                -- instructions concurrentes
        end block label;
```

"guarded" doit être utilisé si l'on désire la synchronisation d'un signal sur la condition de garde

Les blocs (suite)

- **Exemple**

```
entity essai is
-- cette entité n'a pas de signaux
end essai;

architecture test_bench of essai is
-- definition de signaux internes
signal a,b              : bit;
signal horloge,test : bit;
begin
        horloge <= not horloge after 10 ns;
        interne : block ( horloge='1' and not horloge'stable )
                signal s1 : bit;
                begin
                test        <= guarded (a and b) after 2 ns;
                s1          <= (a and b) after 2 ns;
                end block interne;
        a <= '0','1' after 5 ns, '0' after 9 ns, '1' after 15 ns ;
        b <= '1';
end test_bench;
```

L'instruction "generate"

- **Raccourci d'écriture ➠ élaboration itérative ou conditionnelle d'instructions concurrentes**
- **forme conditionnelle :**

➥**les instructions sont générées si la condition est vraie**

```
{label :}  if condition_booléenne generate
           .........
           suite d'instructions concurrentes
           .........
           end generate {label} ;
```

- **forme itérative :**

➥**génération de structures répétitives indicées**

```
{label :}  for nom_de_l'index in intervalle_discret  generate
           ......... -- l'index n'a pas besoin d'être déclaré
           instructions concurrentes
           .........
           end generate {label} ;
```

ex : un buffer tri-state sur 64 bits

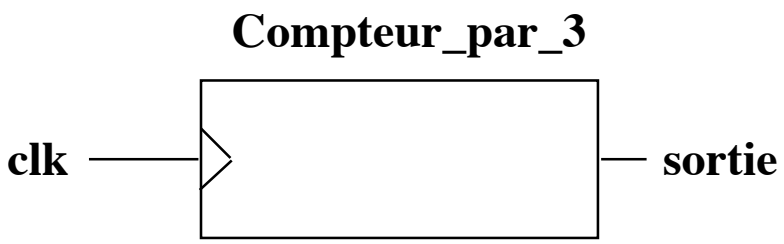
Instructions concurrentes (résumé)

```
architecture concur of modele is
.........
begin
      IC1 : OR_3 port map (A,B,C,OUT);                    -- instanciation de composants

      RAZ : process (reset)                               -- process
                  begin
                        if reset = '1' then S_OUT <= "0000"; end if;
                  end process RAZ;

      A <= (E1 or E2) after 3 ns when ( S = '1') else     -- affectation de signaux
                  '1' after 2 ns;                         -- conditionelle

      interne : block                                     -- bloc
                  begin
                        E1 <= not E1 when ( E2 = '0') else
                                    E2 ;
                  end block interne;
                                                          -- assertion
      assert (S_OUT /= "0000") report "sortie nulle" severity warning;
end coucur;
```

Les instructions séquentielles

- **Elles sont utilisées dans les process ou les corps de sous-programmes**
- **Elles permettent des descriptions comportementales**
  - ➠ **Affectation de variables**
  - ➠ **Affectation séquentielles de signaux**
  - ➠ **Instruction de synchronisation**
  - ➠ **Instructions de contrôle**
  - ➠ **Appels de procédure**
  - ➠ **Assertion**
  - ➠ **Null**

Exemple de process sequentiel




```
Entity compteur_par_3 is
        port ( clk : in bit ;
                sortie : out bit );
end compteur_par_3;
```

```
Architecture sequence of compteur_par_3  is

begin
  calcul : process
        variable etat : integer := 0;
        begin
                wait until clk = '1';
                etat := etat + 1;
                if etat = 3 then
                        sortie <= '1' after 5 ns;
                        wait for 10 ns;
                        sortie <= '0';
                        etat := 0;
                end if;
        end process calcul;
end sequence;
```

Affectation de variables
(Changement immédiat)

```
resul := 4;
resul := resul + 1;
```

Affectation de signal
(affecte une valeur future)

```
S <= 4 after 10 ns;
S2 <= S after 20 ns;
```

structure de test : if then else

```
if condition_booléenne then .....
else ......
end if;                    -- laisser un blanc !!

if condition_booléenn_1 then .....
elsif condition_booléenn_2 then ......
else .......
end if;
```

structure de choix : case

```
case expression is
      when valeur1              =>  ........
      when valeur2|valeur3      =>  ........
      when valeur4 to valeur6   =>  ........
      when others               =>  ........
end case;
```

..... ➜ suite d'instructions séq.

WAIT

```
wait on ... until... for...;
wait on clock, data;
wait on clock until data = '1';
wait on clock until data ='1' for 30 ns;
wait for 10 ns;
wait until clock = '1';
wait;
```

LOOP

```
{label:} while condition loop
......
end loop {label};

{label:} for INDICE  in 1 to 10 loop
......
end loop {label};
-- INDICE n'est pas à déclarer, il est du type integer
      ,n'est pas visible de l'extérieur, pas modifiable
```

```
{label:} loop
......
end loop {label};

next when condition -- saute à l'itération suivante
      ( ou next; -- impératif)
exit when condition -- sort de la boucle
      ( ou exit; -- impératif)
```

ASSERT

```
assert S_OUT /= "0000"
       report "S_OUT est nul "
       severity warning;

-- testé en séquentiel
```

NULL

```
case DIVI is
       when '0' => ..... ;
       when '1' => null;
end case;
```

Appel de procédure

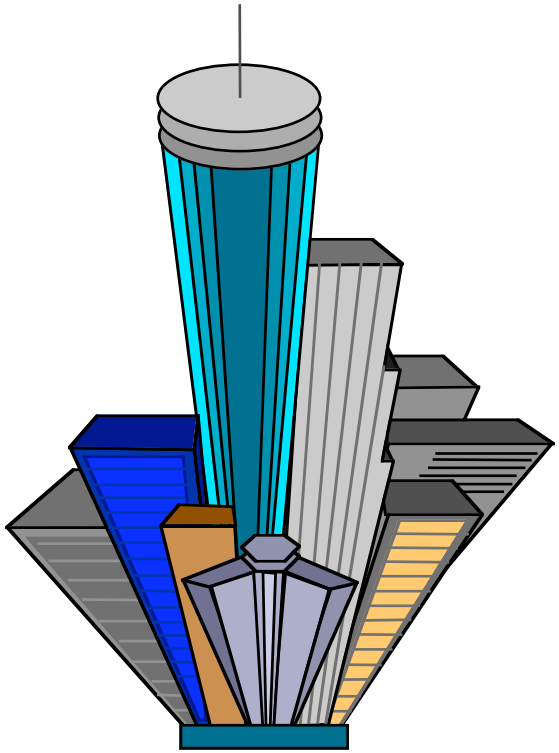
```
       Process
       begin
       ....
       test_valeur (A,B);
       ....
       end Process;

-- Appel  en séquentiel
```

Return

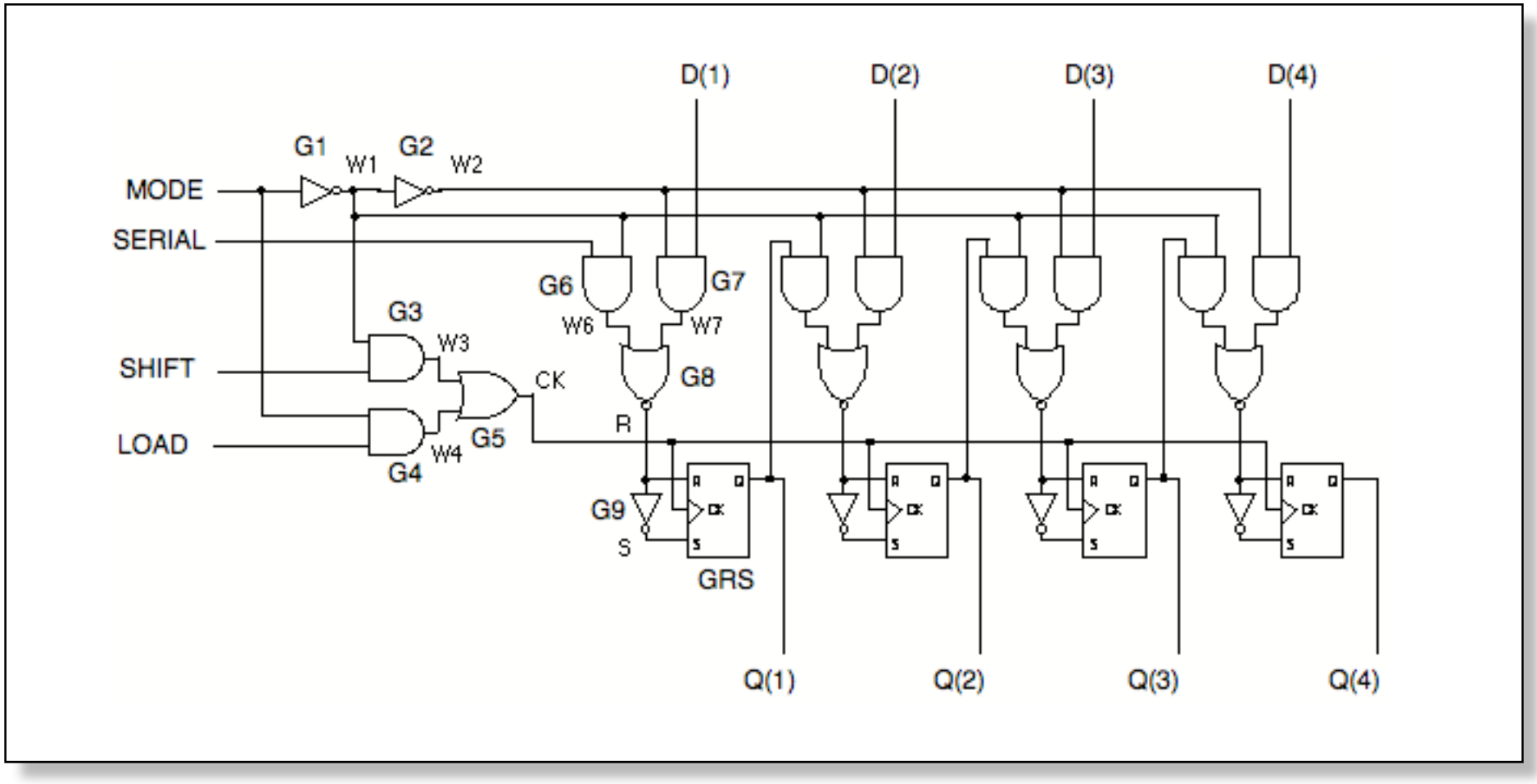
Utilisé dans les sous-programmes pour revenir à l'appelant.

```
return;    -- cas d'une procédure

return valeur; -- cas d'une fonction
```

- **MODELISATION STRUCTURELLE**



REGISTRE à DECALAGE



Description de l'entité

- **interface avec l'extérieure**

```
entity SN_95 is
      port ( SERIAL        : in bit;
                   D       : in bit_vector (1 to 4) ;
               MODE        : in bit;
               SHIFT       : in bit;
               LOAD        : in bit;
                   Q       : inout bit_vector (1 to 4 )) ;
end SN_95
```

✔ Le contenu (architecture) est défini séparément
✔ Une entité peut avoir plusieures architectures

Déclaration des composants

- **Les composants (component) sont des modèles déjà développés et réutilisés dans le nouveau modèle**
- **Le composant doit être déclaré au niveau du modèle :**
  - **dans la partie déclarative de l'architecture**
  - **ou dans un package auquel l'architecture doit faire référence**
- **La déclaration d'un composant spécifie :**
  - **le nom**
  - **le nom des ports**
  - **la direction des données (IN, OUT,INOUT,BUFFER)**
  - **le type de ses ports**
  - **des valeurs par défaut (optionnel)**

Déclaration des composants (suite)

```
Package SN_COMPONENTS is

component INV
      port (E : in bit;
            S : out bit);
end component;

component AND2
      port (E1 : in bit;
            E2 : in bit;
            S : out bit);
end component;

component OR2
      port (E1 : in bit;
            E2 : in bit;
            S : out bit);
end component;

component NOR2
      port (E1 : in bit;
            E2 : in bit;
            S : out bit);
end component;

component RS
      port (R : in bit;
            S : in bit;
            CK : in bit;
            Q : out bit;
            QB: out bit);
end component;

End SN_COMPONENTS;
```

Instanciation des composants

- **Chaque instance est une copie unique du composant avec un nom et une liste de ports**
- **association de ports positionelle**
  - **G3 : AND2 port map (SHIFT,W1,W3);**
- **association de ports nommée**
  - **G3 : AND2 port map (E1 => SHIFT, S => W3, E2 => W1);**
- **open : broche non connectée**
  - **TOTO : AND3 port map (I1,I2,open,S);**
- **un entrée non connectée doit avoir une valeur par défaut**

✔ **les instanciations doivent avoir une étiquette (label)**

Description de l'architecture

```
architecture structurelle of SN_95 is

use WORK.SN_COMPONENTS.all;

signal W1,W2,W3,W4,CK : bit;
signal W6,W7,R,S : bit_vector (1 to 4);

begin
G1 : INV port map (MODE,W1);
G2 : INV port map (W1,W2);
G3 : AND2        port map (SHIFT,W1,W3);
G4 : AND2        port map (LOAD,MODE,W4);
G5 : OR2         port map (W3,W4,CK);

REG : for i in 1 to 4 generate
      PREM : if i = 1 generate
            G6 : AND2 port map (SERIAL, W1,W6(i));
            G7 : AND2 port map (D(i), W2,W7(i));
            G8 : NOR2 port map (W6(i), W7(i),R(i));
```

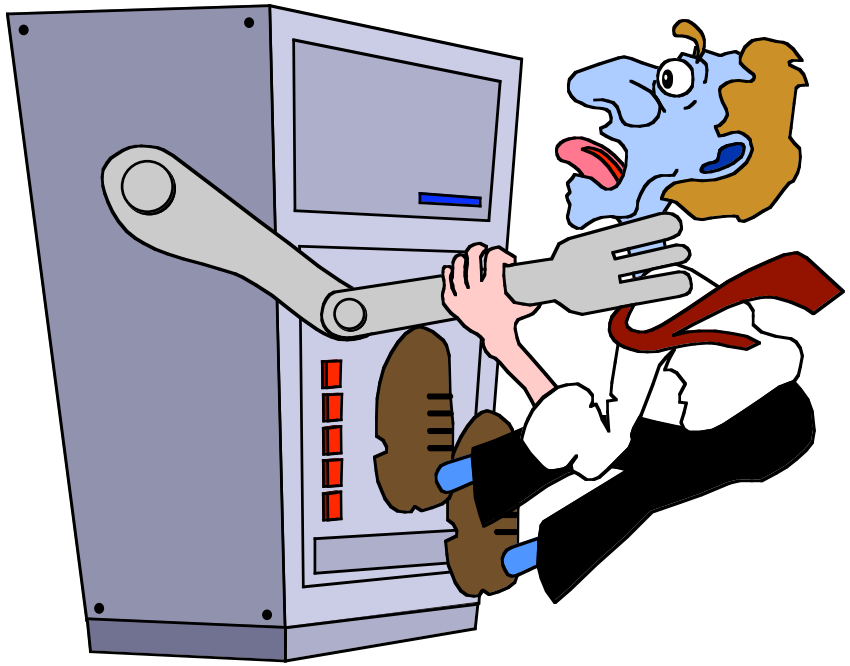
```
        G9 : INV port map (R(i),S(i));
        GRS : RS port map (R(i),S(i),CK,Q(i),open);
    end generate;

    autre : if i/= 1 generate
        G6 : AND2 port map (Q(i-1), W1,W6(i));
        G7 : AND2 port map (D(i), W2,W7(i));
        G8 : NOR2 port map (W6(i), W7(i),R(i));
        G9 : INV port map (R(i),S(i));
        GRS : RS port map (R(i),S(i),CK,Q(i),open);
    end generate

end generate

end structurelle;
```

Configuration du système

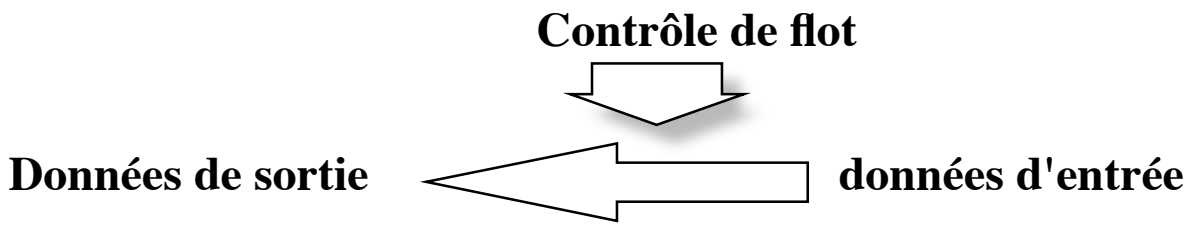
- **Elle permet d'associer à chaque instance, une entité et une architecture**

```
library TTL;
configuration config_1 of SN_95 is
    for structurelle
        for all : AND2 use entity TTL.AND2(FAST); end for;
        for all : OR2 use entity TTL.OR2(FAST); end for;
        for G1,G2 : INV use entity TTL.INVERTER(MU07); end for;
        for REG
            for all : NOR2 use entity TTL.NOR2(FAST); end for;
            for all : RS use entity TTL.BASC_RS(BEH); end for;
            for PREM
                for G9 : INV use entity  TTL.INVERTER(FAST); end for;
            end for;
            for AUTRE
                for G9 : INV use entity  WORK.INVERSEUR(le_mien); end for;
            end for;
        end for;
    end structurelle;
```

- **MODELISATION : FLOT de DONNEES**



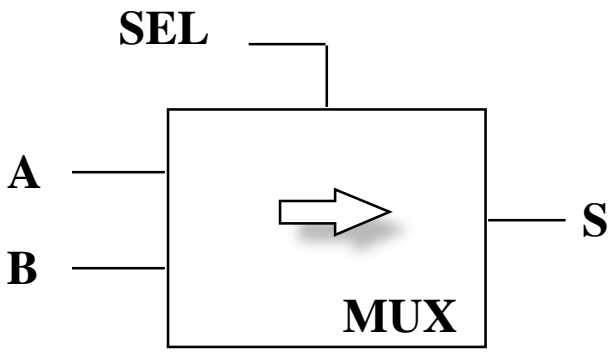
Circulation de données



- **Utilise les instructions concurrentes d'affectation de signal**
  - **affectation simple** — **a<= b + c after 10 ns;**
  - **affectation sur condition** — **a <= '0' when .....**
  - **affectation par sélection** — **with expression select.....**
- **les formes d'ondes sont de même type que la cible**
- **aucun signal à droite ne peut être en mode "out"**
- **La cible ne peut être qu'un objet signal**

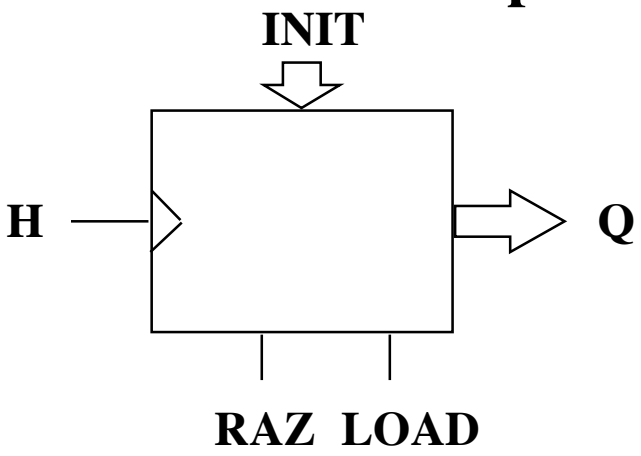
Affectation sur condition (exemple)

- **Cas d'un multiplexeur**




```
S <= A after 5 ns when SEL = '1' else
     B after 4 ns;
```

- **Cas d'un compteur modulo 12**

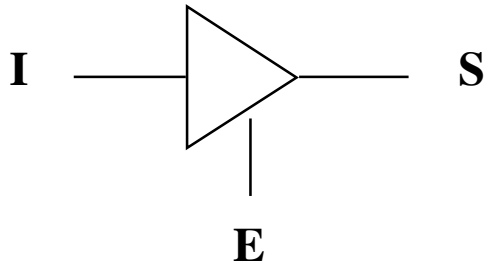



```
Q <= 0 when RAZ = '1' or Q = 12 else
      INIT when LOAD ='1' else
     Q+1 when (H='1' and H'event) else
     Q ;
```

✔ les conditions sont évaluées dans l'ordre
✔ la dernière est celle par défaut

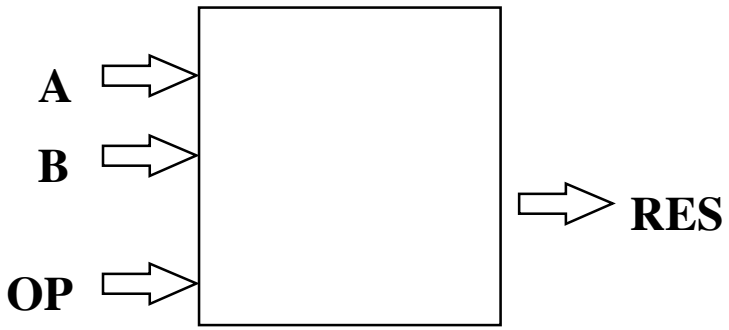
Affectation par sélection (exemple)

- **Buffer 3 états**




```
with E select
      S <=  I when '1',
            'Z' when '0',
            'X' when others;
```

- **ALU**

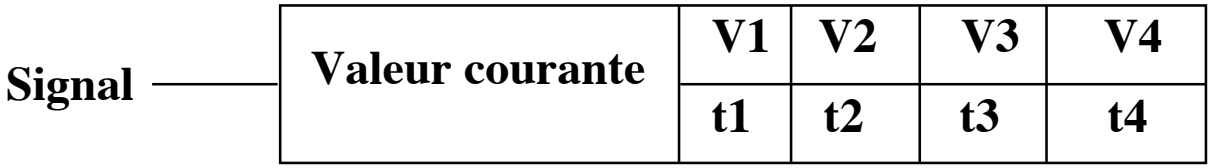



```
with OP select
      RES <= A and B when "0000",
             A or B when "0001",
             not (A and B) when "0010",
             .....
             A xor B when "1111";
```

✔ le driver est réévalué si modification de la forme d'onde ou du sélecteur
✔ toutes les possibilités sont prévues

Affectation de signal

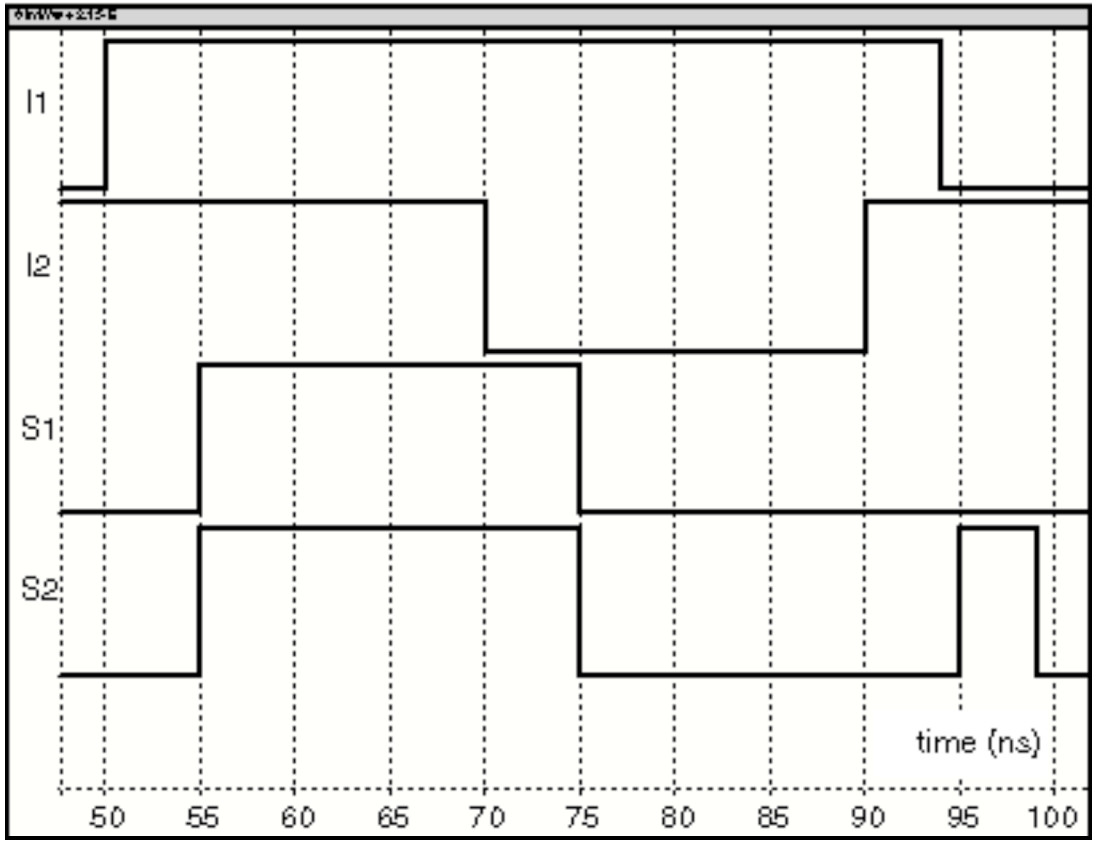
- **A chaque signal est associé un "driver"**



Il contient la valeur du signal à l'instant présent ansi que la liste des futures valeurs déjà calculées

- **Le driver est recalculé à chaque modification de la forme d'onde**
- **Délai nul -> pas temporel du simulateur**

Driver et filtrage



- **S1 <= I1 and I2 after 5ns; -- (modèle inertiel)**
- **S2 <= transport I1 and I2 after 5 ns; -- (modèle transfert)**

  ➠ **modèle inertiel filtre les impulsions < temps de transmission**

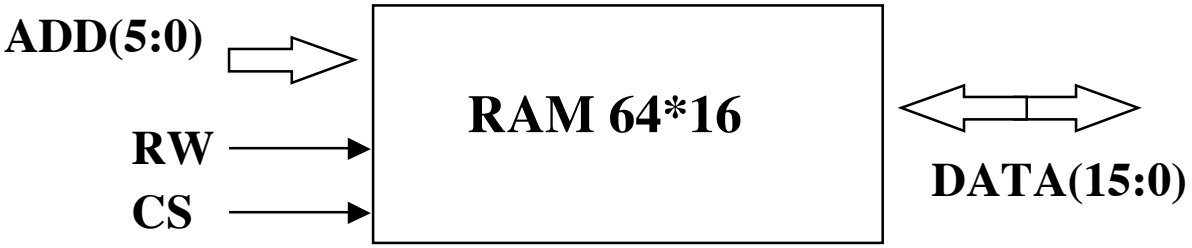
  ➠ **transport : évite le filtrage**

MODELISATION COMPORTEMENTALE

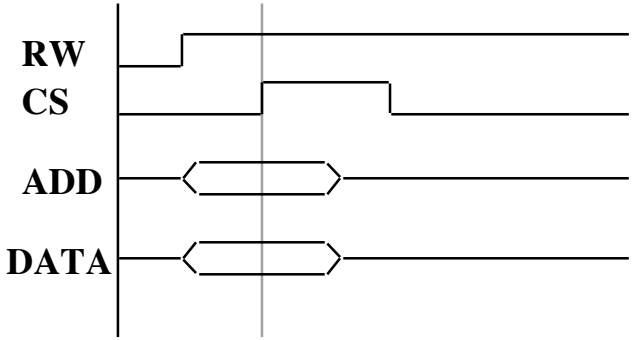
Description comportementale

- **Modélisation algorithmique**
- **Pas de relation avec l'implantation du système**
- **Utilise un langage séquentiel, structuré et de haut niveau**
- **On retrouve cette description dans 2 unités :**
  - » **les process**
  - » **les sous-programmes**

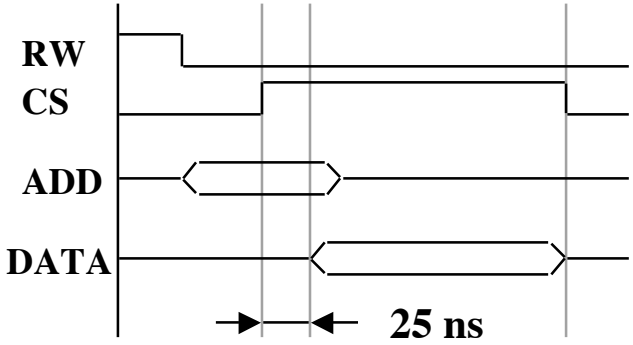
Exemples de process



•**Ecriture dans la mémoire**



•**Lecture de la mémoire**



```
-- description de l'entite
entity Ram is
        Port (  ADD : in std_logic_vector (5 downto 0); -- bus d'adresse
                RW: in Bit;
                CS: in bit;
                DATA: inout std_logic_vector (15 downto 0));
end Ram;

Architecture Comp of ram is
        Subtype MOT is std_logic_vector (15 downto 0);
        Type MATRIX is array (0 to 63) of MOT;
        begin
                Process (CS)
                 variable MEMOIRE:Matrix;
                begin
                if CS ='1' then
                        if RW='1' then
                                MEMOIRE(TO_INTEGER(ADD)):=DATA;
                        else DATA <= MEMOIRE(TO_INTEGER(ADD)) after 25 ns;
                    end if;
                else DATA <= (others => 'Z');
                End if;
        End process;
End Comp;
```

Fonctions et procédures

- **Les procédures admettent des paramètres d'entrée et de sortie**
  - » **calcul_resul (a,b,20);**
- **Les fonctions admettent des paramètres d'entrée et retourne une seule valeur**
  - » **data := autre_data and plus_petit(a,b);**
- **Les sous-programme sont définis en deux temps**
  - **la partie déclarative (elle est optionelle et n'est utilisée que dans les packages)**
    - » **procedure nom_procédure (liste_paramètres);**
    - » **function nom_function (liste_paramètres) return type_resultat;**
  - **le corps du sous-programme**

```
procedure nom_procédure ( liste_paramètres) is
( ou function nom_function (liste_paramètres) return type_resultat is )
{ déclarations locales}
begin
.....            -- return valeur (cas de la fonction);
end;
```

Fonctions et procédures (suite)

- **Les sous-programmes peuvent être définis :**
  - **dans la partie déclarative d'un process**
  - **dans la partie déclarative d'un autre sous-programme**
  - **dans un package**
- **la liste des paramètres est de la forme:**
  - **{objet} nom1, nom2.. : {mode} type {:= valeur par défaut}**

	Constantes	Variables	Signaux
in (procédures)	Autorisé	Autorisé	Autorisé
out (procédures)		Autorisé	Autorisé
inout (procédures)		Autorisé	Autorisé
in (fonctions)	Autorisé		Autorisé
out (fonctions)			
inout (fonctions)			

Fonctions et procédures (exemples)

```
function MIN (A,B : INTEGER) return INTEGER is
begin
        if A<B then
                return A;
        else
                return B;
        end if;
end MIN;
```

```
procedure ajoute ( variable a : inout real,
                                b : in real := 1.0) is
begin
        a := a + b;
end ajoute;
```

Les surcharges

- **Deux sous-programmes sont surchargés si ils ont le même nom mais une liste des paramètres différente**

  exemple :

  ```
  procedure CONVERSION ( entree : in real, sortie : out integer);
  procedure CONVERSION ( entree : in bit, sortie : out integer);
                                        ou
  function MIN (A,B : integer ) return integer;
  function MIN (A,B : bit ) return bit;
  ```

- **Il est possible de surcharger les opérateurs prédéfinis**

  exemple :

  ```
  function "AND" (A,B : feu_tricolore) return boolean;    -- feu1 and feu2
  function "+" (A,B : bit_vector) return bit_vector;
  ```

Exemples de surcharge

```
type feu_tricolore is (vert,orange,rouge,panne);
.............
function "+" ( A,B : feu_tricolore ) return feu_tricolore is

type matrice is array (feu_tricolore range <>,feu_tricolore range <>)
                    of feu_tricolore;
constant carte : matrice := -- vert orange rouge panne
                          (( vert, orange, rouge, panne ),          -- vert
                          ( orange, orange, rouge, panne ),         -- orange
                          ( rouge, rouge, rouge, panne ),           -- rouge
                          ( panne , panne , panne , panne ));       -- panne
begin
      return carte (A,B);
end "+";
```

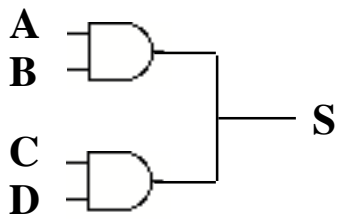
Les signaux à sources multiples

- **Si un signal se trouve à gauche de plusieurs affectations il est dit multi-sources**

ex : **S <= A and B;**
S <= C and D;

S = ????

➥ **erreur à la compilation !!**



- **On doit définir alors une fonction de résolution associée à ce type de signal**
  ➠ **gestion des conflits**

✓ **Il n'existe pas de fonction de résolution par défaut !!**

Exemple de fonction de résolution

```
Package fonction_de_resolution is
      function et_resolu ( sources : bit_vector)
                        return bit;
      subtype bit_resolu is et_resolu bit;
end fonction_de_resolution;

Package body fonction_de_resolution is
      function et_resolu ( sources : bit_vector)
                        return bit is
            begin
            for I in sources'range loop
                  if sources(i) = '0' then return '0'
                  end if;
            end loop;
            return '1';
      end et_resolu;
end fonction_de_resolution;
```

```
use work.fonction_de_resolution.all;
entity test is
      port (a,b,c,d : in bit_resolu;
            e : out bit_resolu);
end test;

architecture nico of test is
begin

e <= a or b after 15 ns;
e <= c and d after 15 ns;
e <= a or d after 20 ns;

end nico;
```

Les packages

- **Utilisés pour mettre des éléments à disposition de plusieurs modèles**
- **Seules les ressources suivantes sont autorisées :**
  - **déclarations de type (ou sous-types)**
  - **déclarations d'objets (sauf variables);**
  - **déclarations de composants**
  - **déclarations et corps de sous programmes**
- **Ils possédent**
  - **une partie déclarative**
  - **le corps du package (package body)**
- **On utilise la clause use pour utiliser les éléments du package**
  - **use librarie.nom_du_package.nom_de_la_ressource;**

Exemple de packages

```
Package geometrie is

constant pi :real := 3.141592654;
procedure aire_cercle (  a: in real ;  b: out real);

end geometrie;


package body geometrie is

procedure aire_cercle (  a: in real ;  b: out real) is
            begin
            b := pi * a**2 ;
            end;
end geometrie;
```

```
use work.geometrie.pi;
use work.geometrie.aire_cercle;

.........
calc: process(h)
variable aire, perim :real ;
begin
      .....
      aire_cercle (rayon,aire);
      perim := 2*pi*rayon;
      .....
end process calc;
.......
```