VHDL

Support de cours

D.Chillet

Daniel.Chillet@enssat.fr

http://r2d2.enssat.fr

Qu'est ce que ça veut dire ?

Vhsic **H**ardware **D**escription **L**anguage

Vhsic : Very High Speed Integrated Circuit

Langage de description de systèmes matériels

Plan

1) Introduction : motivations, historique

2) Méthode de conception: modèles de description

3) Unités de conception :

entité, architecture, configuration, paquetage, corps de paquetage

4) Les objets : signaux, types, sous types

5) Les éléments du langage : opérateurs, procédures, fonctions, affectations

6) Instructions concurrentes : processus, blocs

7) La généricité

8) La fonction de résolution

9) Simulation et validation, réalisation d'un composant de test

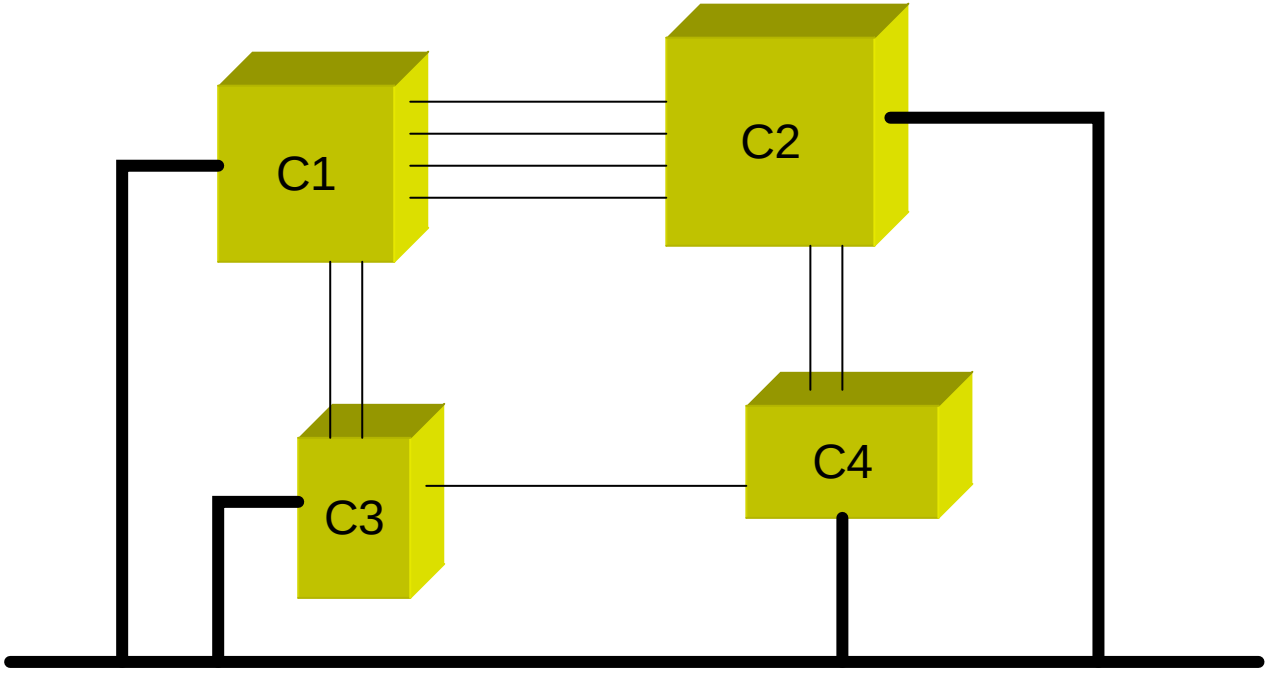
10) Exemples : paquetage standard et exemples de codes VHDL

Introduction

1) Introduction

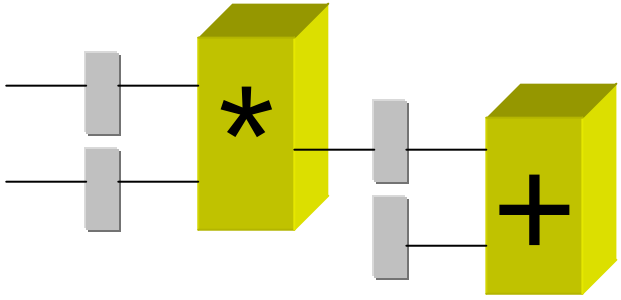
Un HDL, qu'est ce que c'est ?

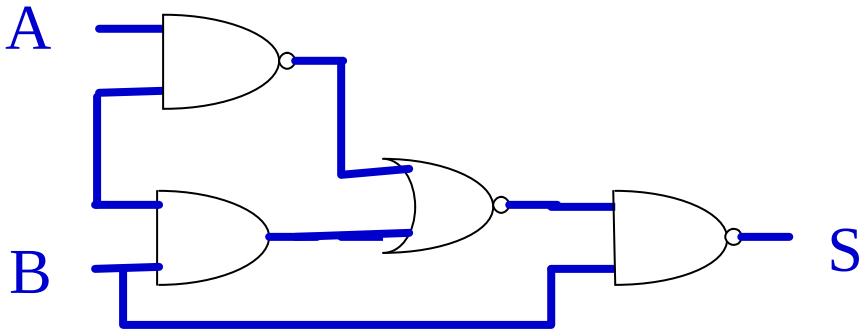
- moyen de décrire un **système matériel** :
  - ✓ qu'est qu'un système matériel ?
    - en général, il s'agit d'un schéma mettant en œuvre :
      - un certain nombre de composants
      - des connexions entre composants

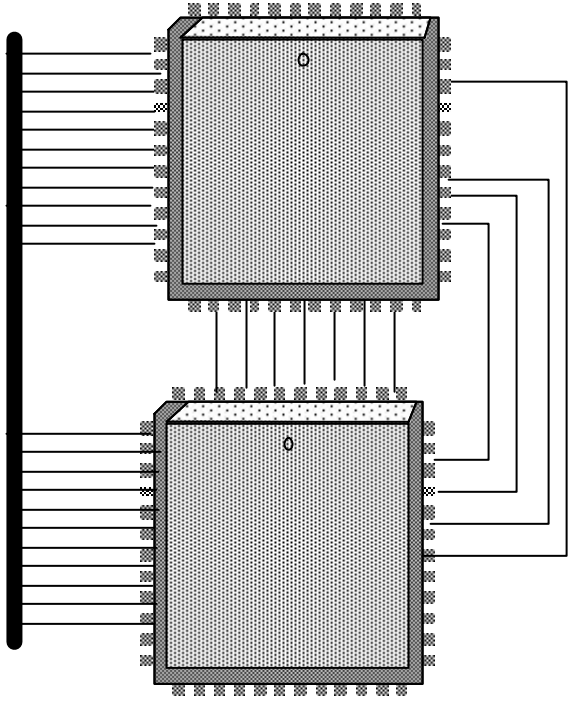


1) Introduction

- les niveaux de description peuvent être variables :
  - ✓ niveau logique
  - ✓ niveau transfert de registres
  - ✓ niveau système





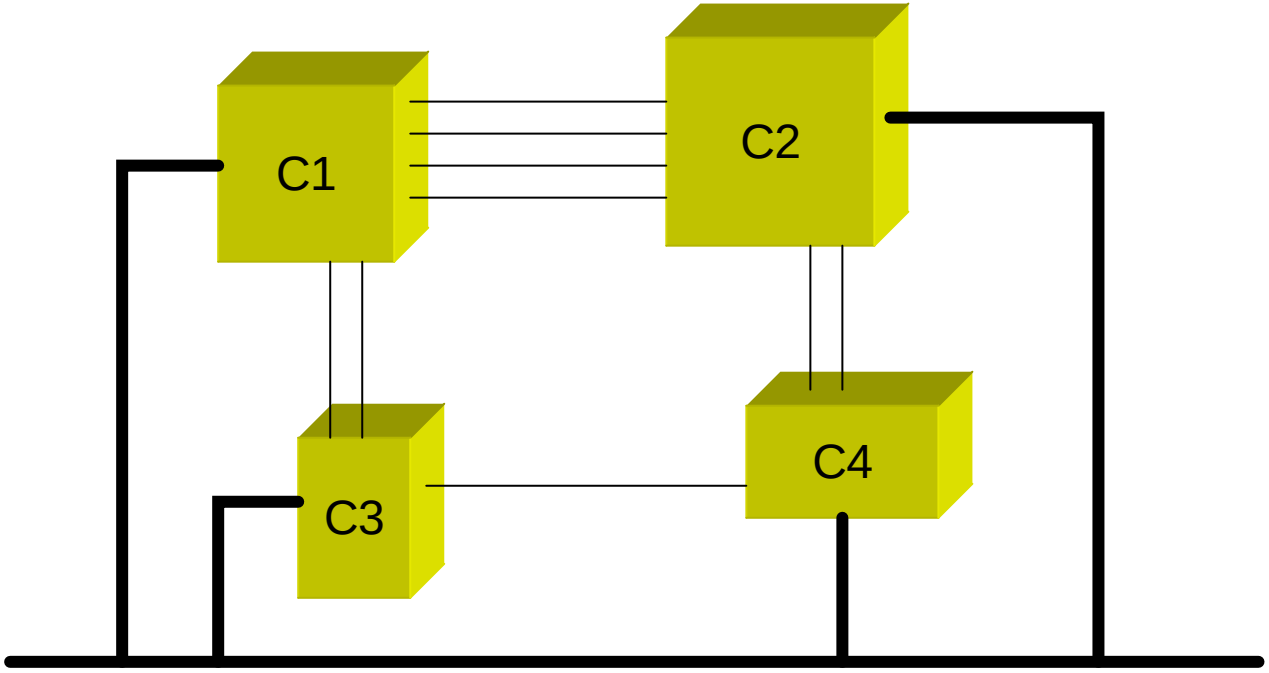


1) Introduction

- Un HDL, doit :
  - permettre la modélisation du système :
    - ✓ formalisation du problème
    - ✓ à partir de briques de base
    - ✓ éviter les descriptions ambiguë
  - faciliter la documentation du système :
    - ✓ VHDL est très verbeux !!!

1) Introduction

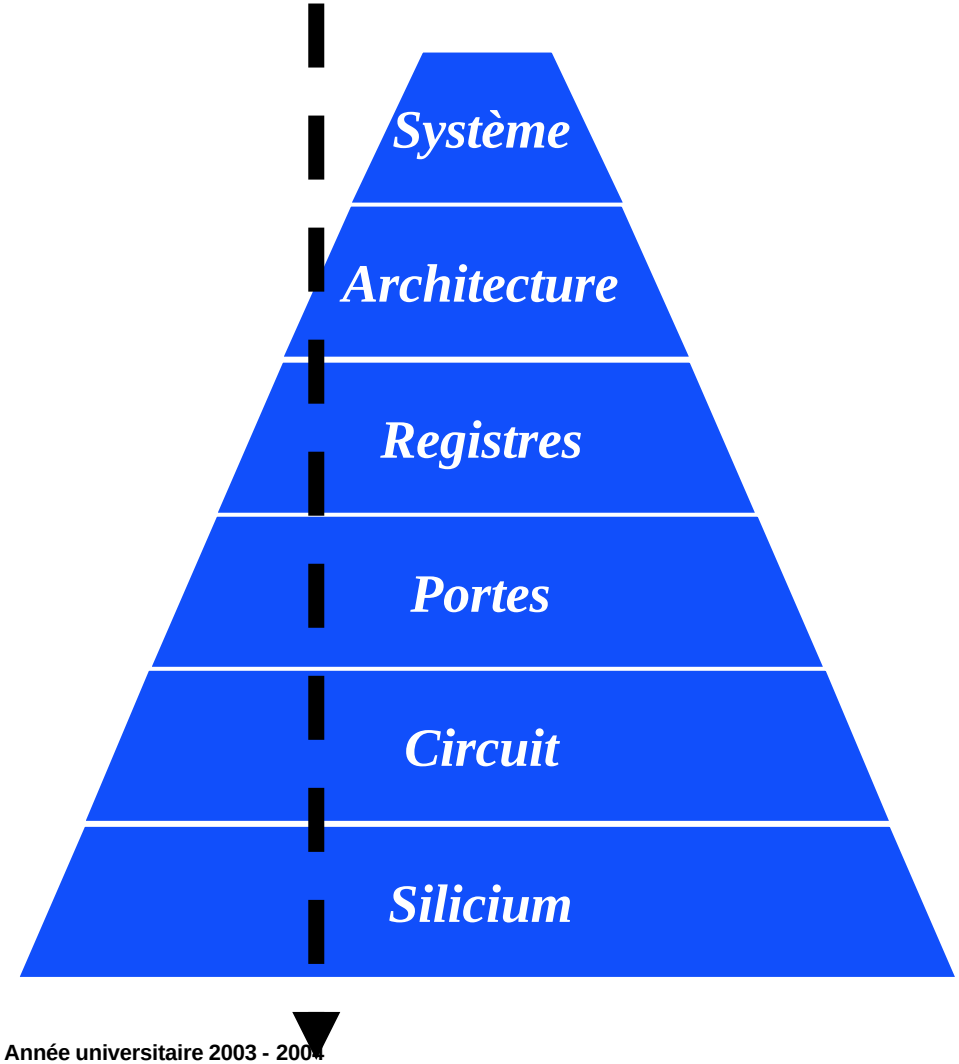
- la validation du système :
  - ✓ prouver le fonctionnement du système décrit
  - ✓ simulation :



- une structure schématique ne peut être simulée que si on connaît le ***comportement*** de chacun des sous systèmes

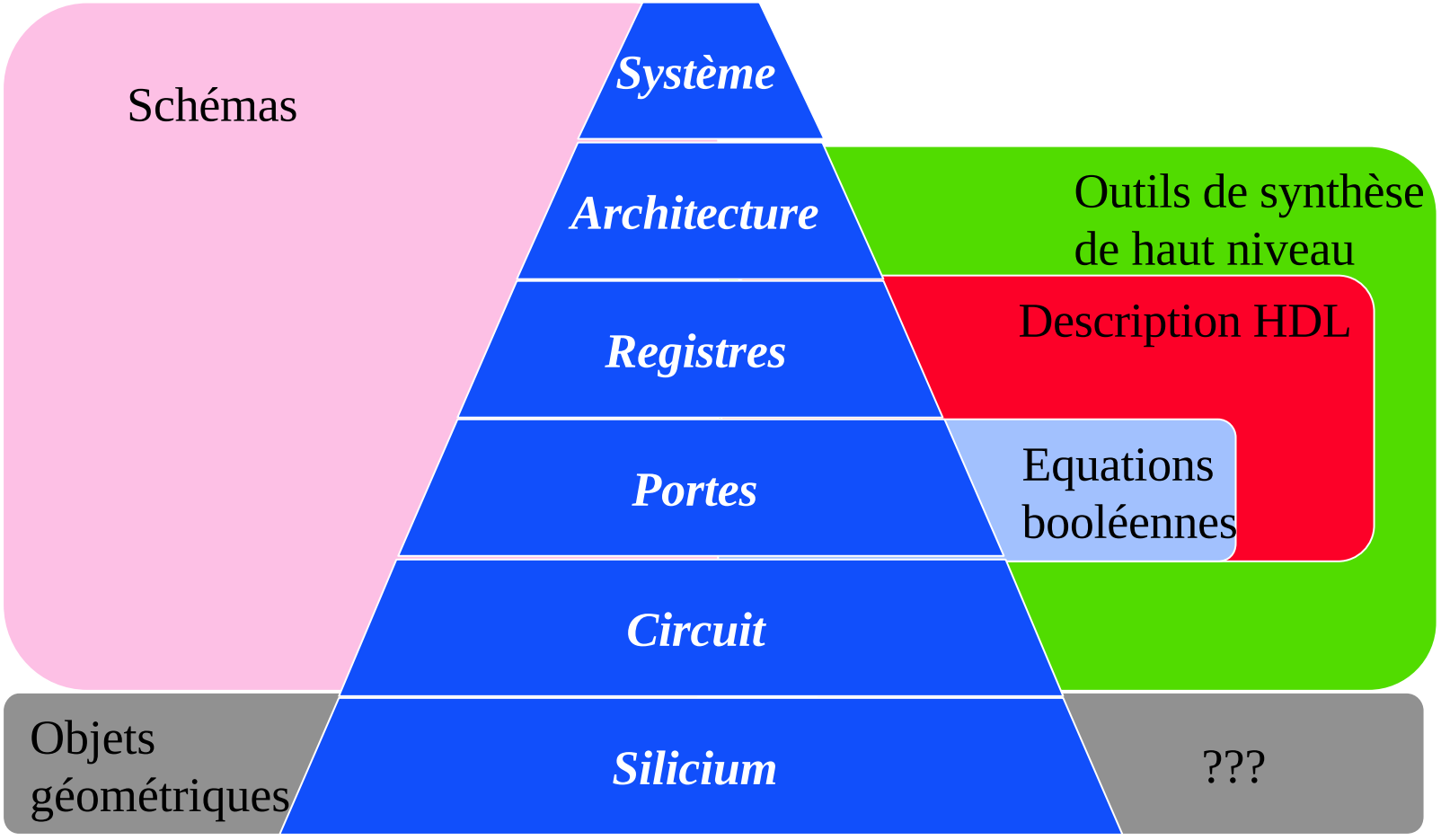
1) Introduction

- suivre la conception du système :
  - ✓ si possible pas de changement de langage à chaque étape de conception

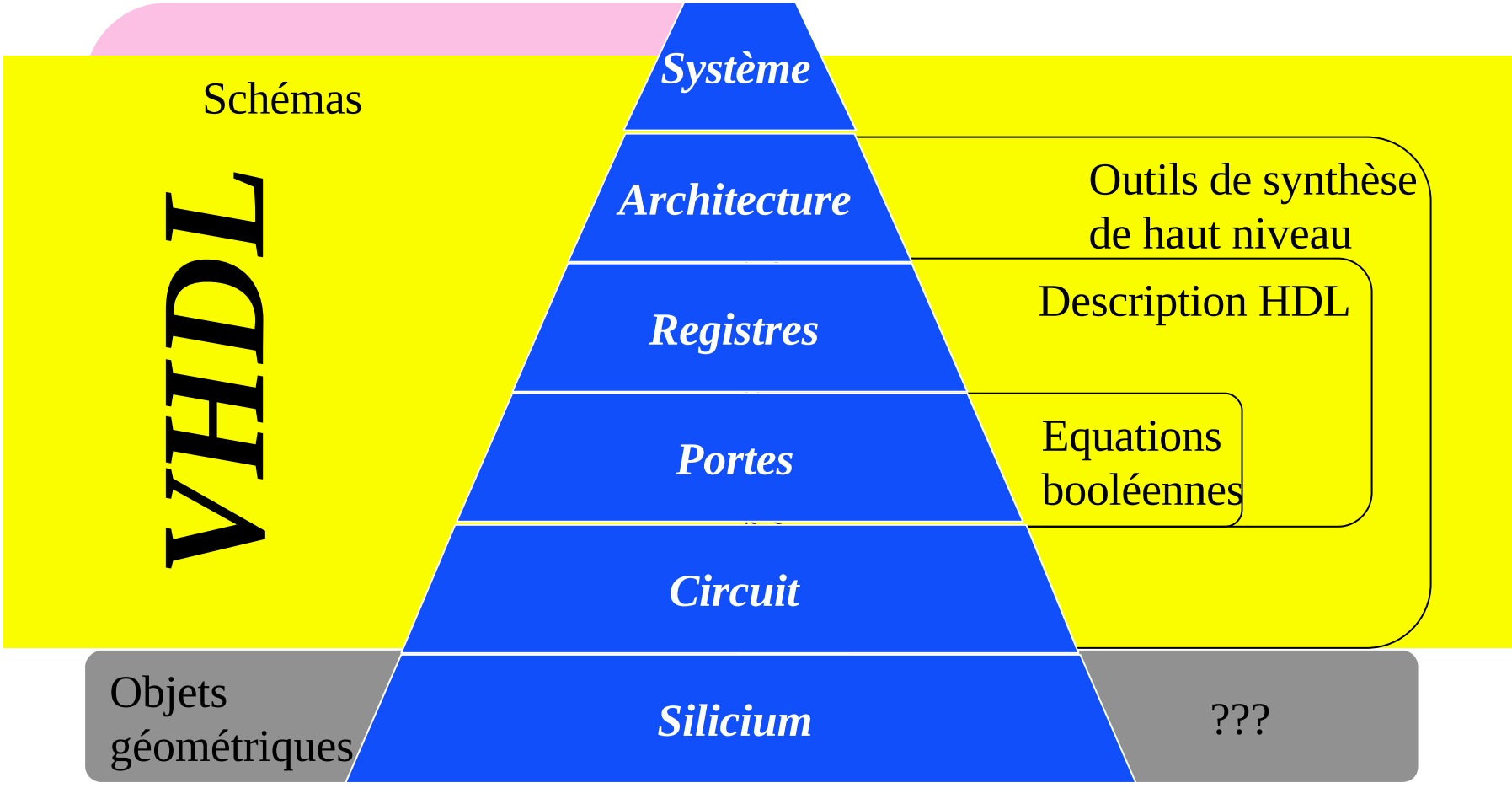


1) Introduction

- ✓ Outils et objets manipulés



1) Introduction



1) Introduction

- Motivations pour une approche méthodologique :
  - évolutions technologiques rapprochées :
    - ✓ la durée de vie des équipements est supérieure au laps de temps entre 2 technologies :
      - donc les systèmes doivent pouvoir supporter une évolution technologique
    - ✓ se lier à une technologie dès la phase de conception peut conduire à mettre sur le marché un produit déjà dépassé
  - indépendance vis à vis des fournisseurs :
    - ✓ se lier à un fournisseur dès la phase de conception peut conduire à revoir toute la conception si les composants ne sont plus disponibles
  - besoin de standardisation
  - nécessité de moyen de description non ambiguë des systèmes matériel :
    - ✓ début des années 80

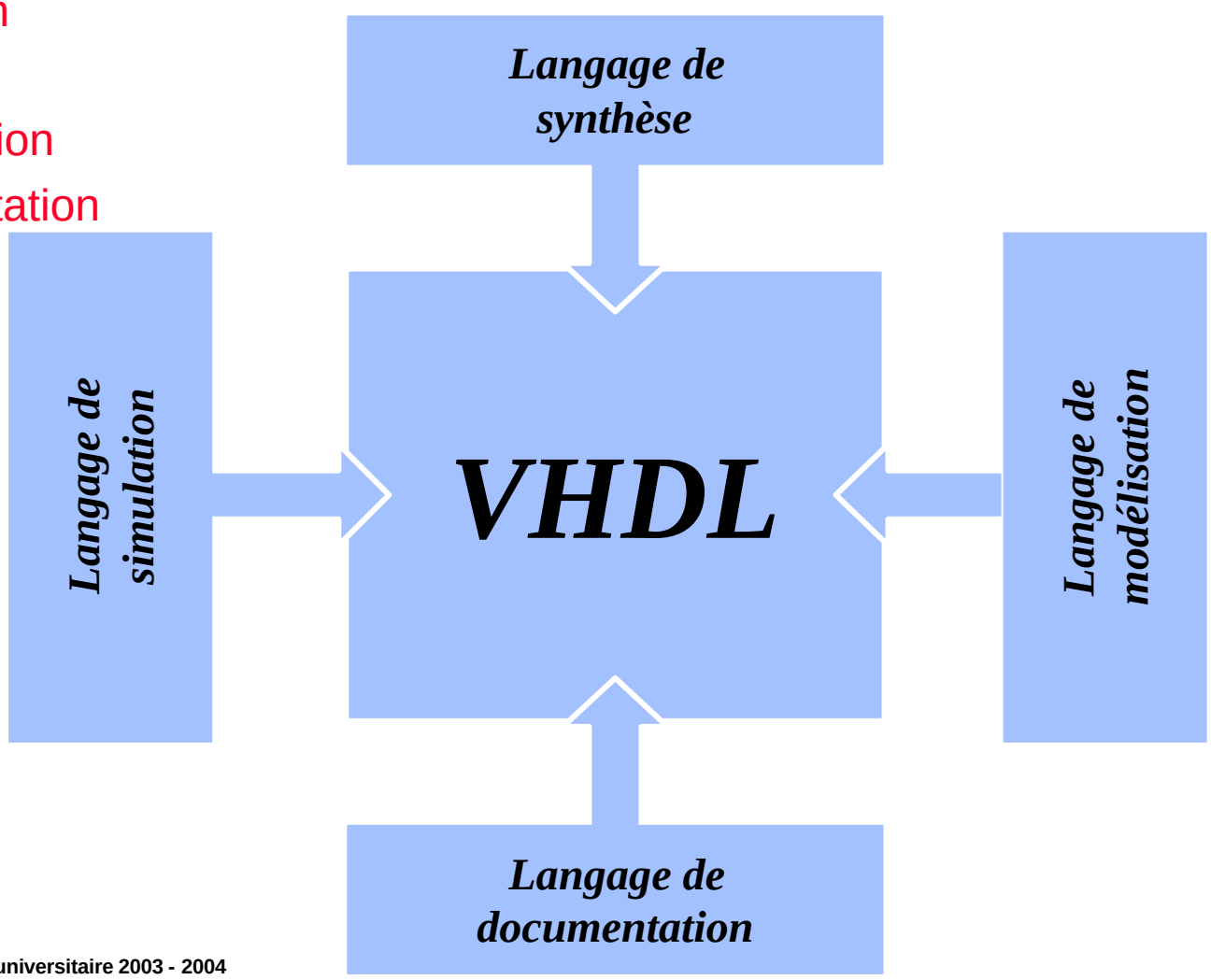
1) Introduction

Rapide historique :

- DOD en 81: effort de normalisation entre 83 et 87
  - ✓ département de la défense Américaine
- Normalisation IEEE en 87 (IEEE 1076) :
  - ✓ efforts de normalisation : Intermetrics, IBM, Texas instrument
- Le langage est proche de ADA :
  - ✓ références constantes
  - ✓ ADA a été normalisé définitivement en 83
- Nouvelle norme en 93
- Nouvelle normalisation autour de VHDL analogique :
  - ✓ VHDL AMS
  - ✓ pourquoi ?
    - demande en simulation analogique et en simulation mixte

1) Introduction

- VHDL :
  - ne vise pas une exécution, il est utilisé pour :
    - ✓ la simulation
    - ✓ la synthèse
    - ✓ la spécification
    - ✓ la documentation



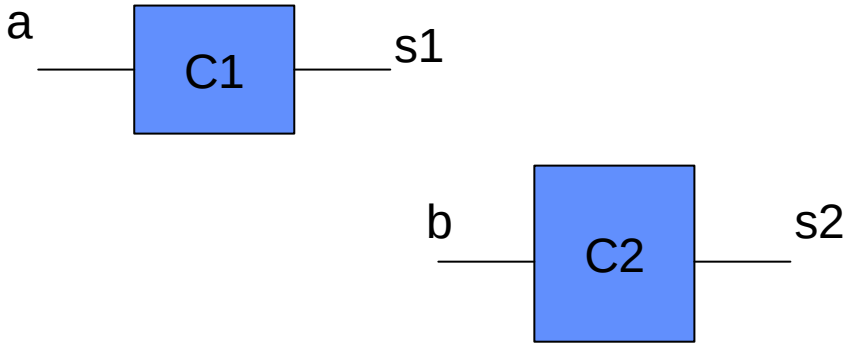
1) Introduction

- la brique de base est le composant ou l'entité :
  - ✓ tout système est bâti à partir de cette brique.
  - ✓ dans les langages de programmation les briques de base sont les procédures et les fonctions :
    - la procédure est appelée puis oubliée :
      - activation explicite
    - le composant existe en soi :
      - activation implicite par événements sur ses entrées

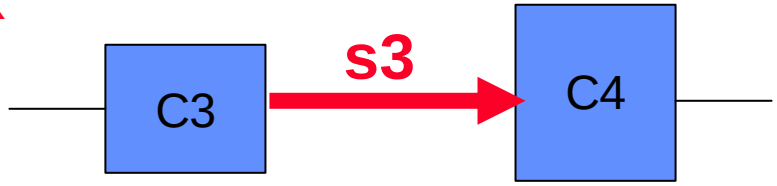
```
main()
{
        ...
        resultat = Factorielle(x)
        ...
}
```

1) Introduction

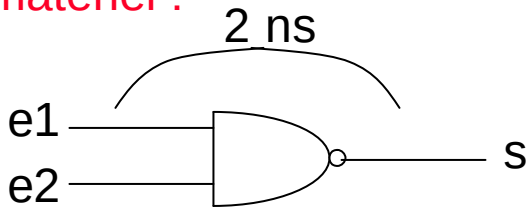
- fonctionnement concurrent des composants :
  - ✓ si a et b évoluent en même temps, alors les composants c1 et c2 ont un fonctionnement parallèle (concurrent)



- notion de signal :
  - ✓ connexion des composants entre eux

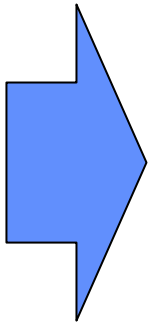


- la notion de temps est gérée :
  - ✓ prise en compte des contraintes réel du matériel :
    - temps de traversée,
    - de calcul,
    - etc.



1) Introduction

- Les avantages de VHDL :
  - indépendant du constructeur
  - indépendant de la technologie
  - indépendant de la démarche
  - indépendant du niveau de conception

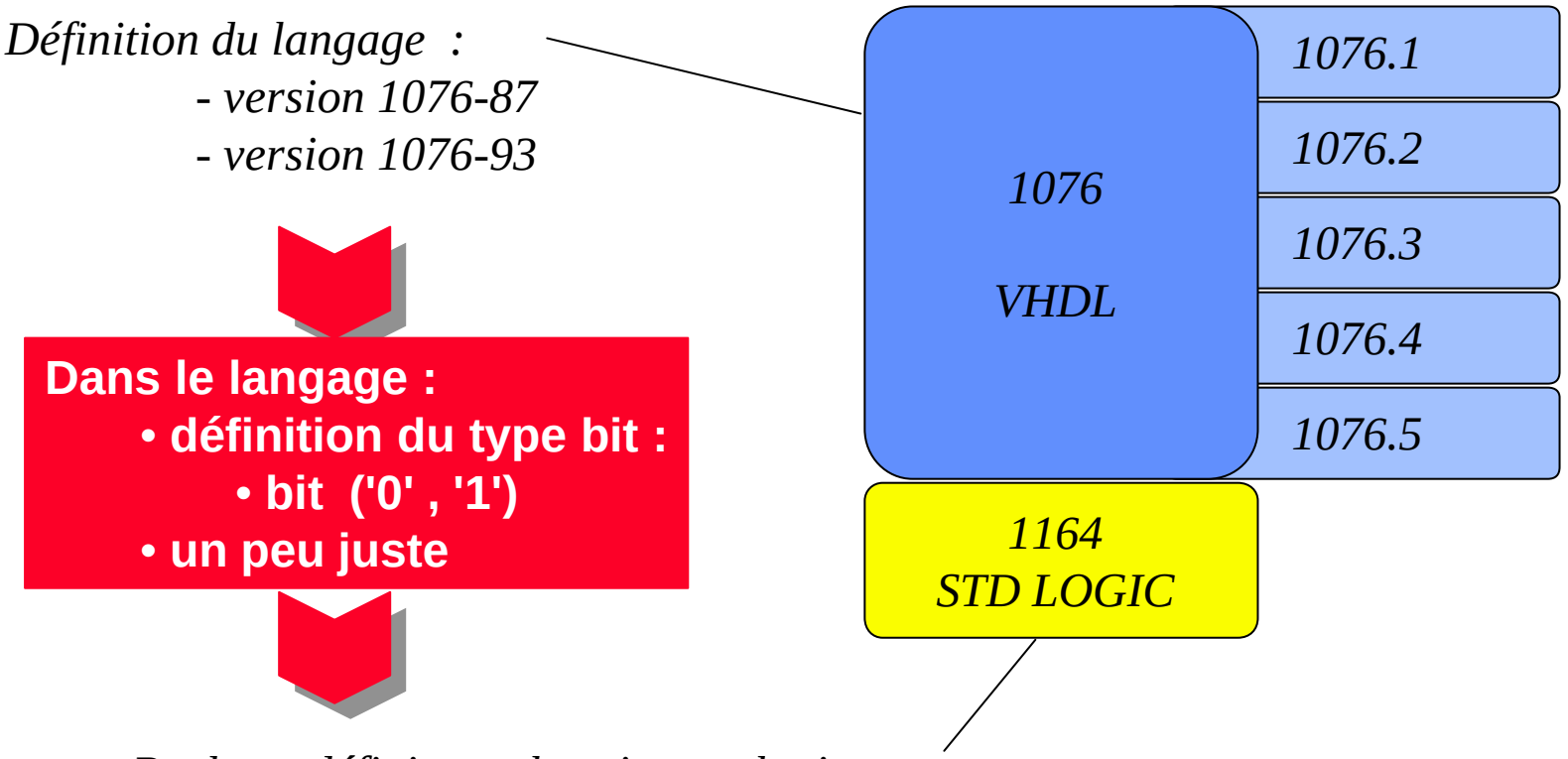


Portabilité

  - **Standard IEEE :**
    - ✓ reconnu par les vendeurs d'outils CAO
    - ✓ grand nombres de bibliothèques :
      - d'opérateurs
      - de composants
      - de fonction
    - ✓ nombreux outils :
      - compilateur, simulateur
      - mais aussi des outils de synthèse et de spécification

1) Introduction

Définition du langage :
- version 1076-87
- version 1076-93



Package définissant les niveaux logique
nécessaire pour la description des systèmes électronique

Utilisé par tous les constructeurs
et par tous les outils

1) Introduction

- VHDL :
  - Langage moderne, puissant, général :
    - ✓ jeu d'instructions complet et très riche
    - ✓ fort typage des données
    - ✓ compilation séparée des entités
- Développement parallèle facilité :
  - travail d'équipe :
    - ✓ découpage en unités de conception
    - ✓ développement et validation parallèle
    - ✓ nécessite de bien identifier les interfaces entre les blocs fonctionnels
- Simulation du système :
  - à tous les niveaux :
    - ✓ cahier des charges !!!
    - ✓ modèles
    - ✓ réalisation

Erreurs découvertes au plus tôt dans le cycle de conception

1) Introduction

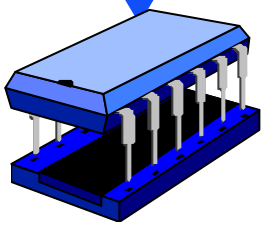
1 ère année :
- modélisation
- simulation

Description informelle

Spécification en VHDL

Synthèse

2 ème année :
- modélisation
- simulation
- synthèse



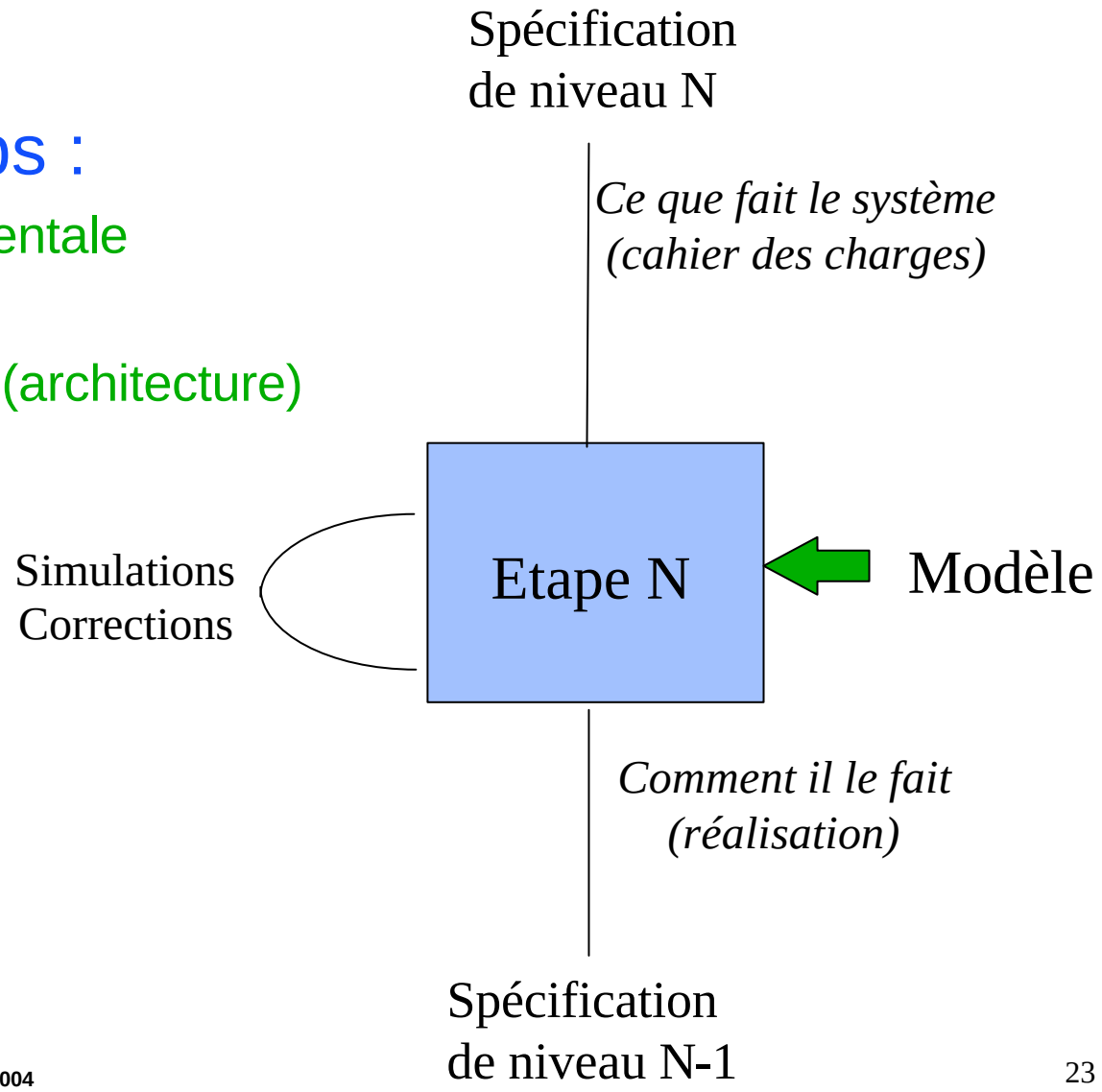
Méthodes de conception

2) Méthodes de conception

- Descendante :
  - on part du besoin et descend jusqu'au niveau physique
  - généralement, on s'arrête au niveau d'une bibliothèque :
    - ✓ d'opérateurs logiques
    - ✓ d'opérateurs arithmétiques
    - ✓ d'opérateurs plus complexes
- Ascendante :
  - on part d'une bibliothèque pour remonter, par assemblage, au système à concevoir
  - la bibliothèque est décrite à un niveau:
    - ✓ opérateurs logiques
    - ✓ opérateurs arithmétiques
    - ✓ opérateurs plus complexes

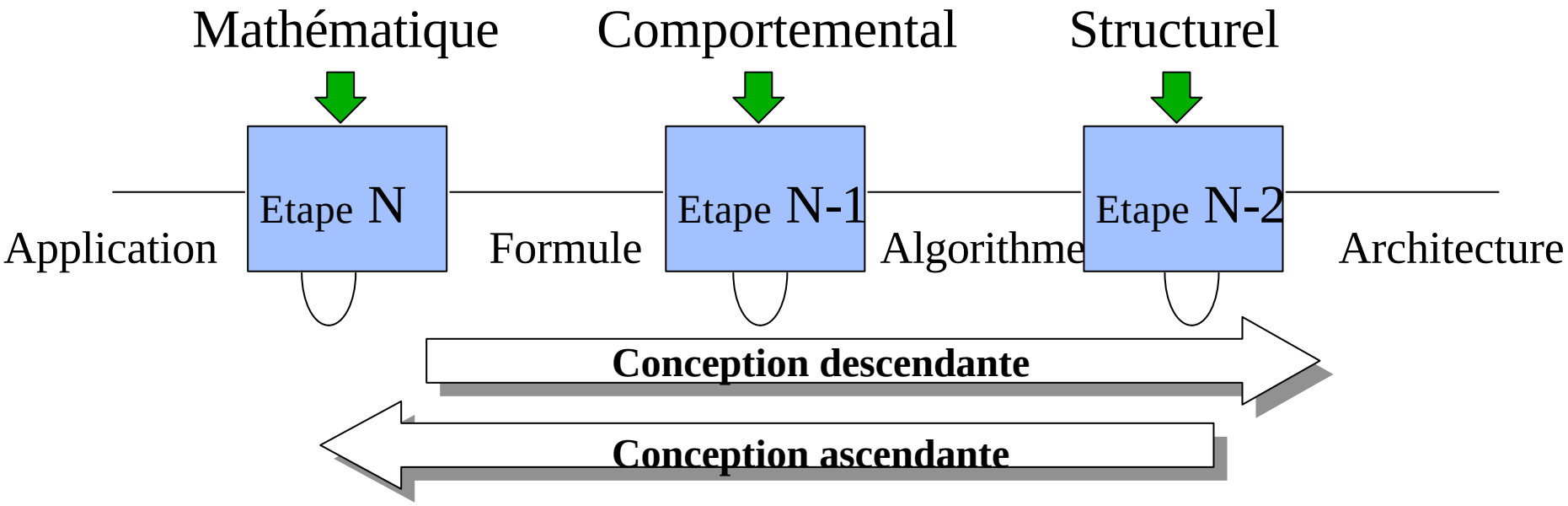
2) Méthodes de conception

- Conception descendante : dédiée
  - concevoir vite, bien
- Méthode en 3 temps :
  - description comportementale
  - simulation et validation
  - description structurelle (architecture)



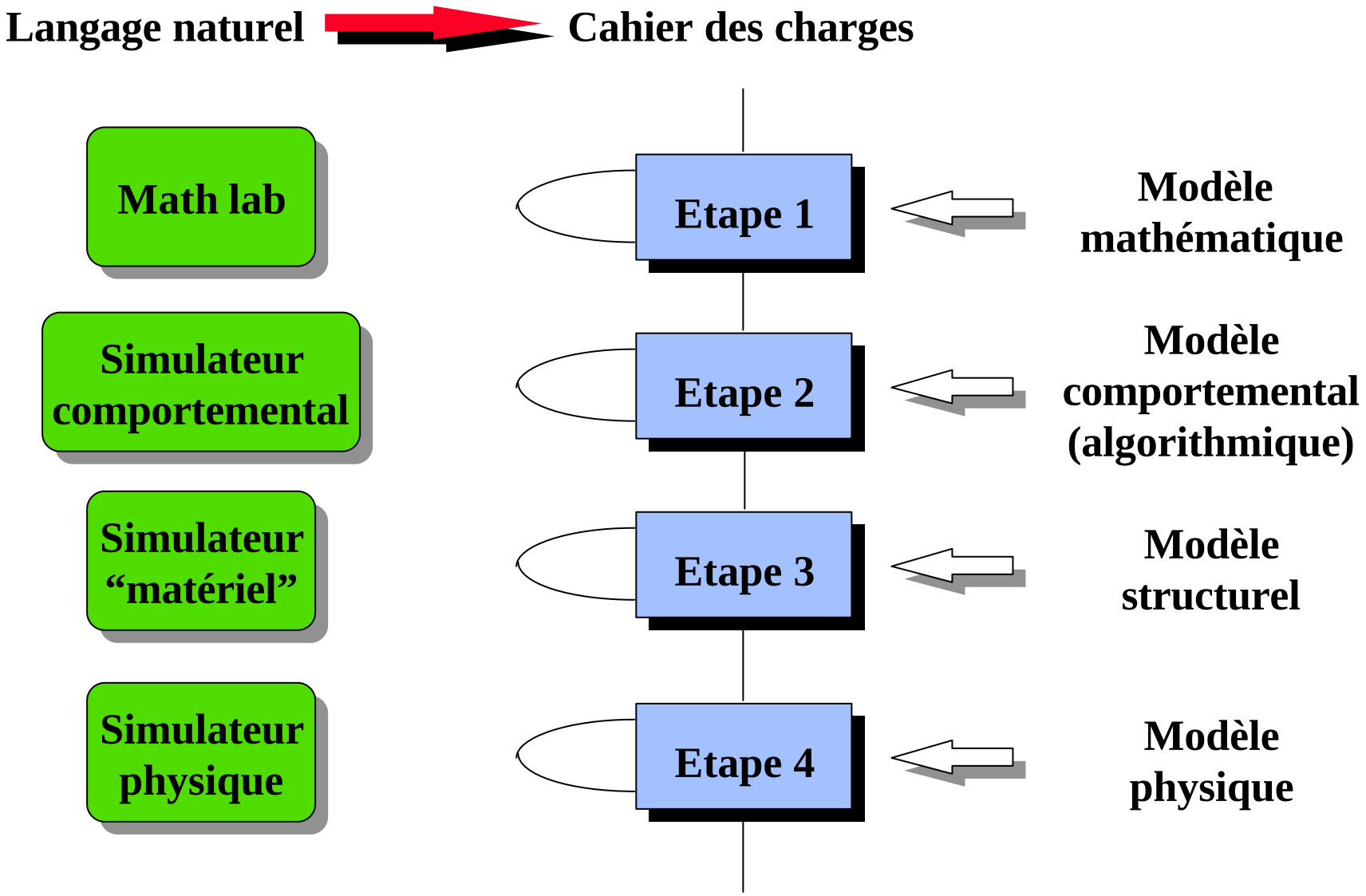
2) Méthodes de conception

- Raffinement successifs des descriptions (hiérarchisation)
  - un bloc est définit comme un assemblage de sous blocs reliés entre eux par des signaux
  - garantie l'équivalence entre les niveaux

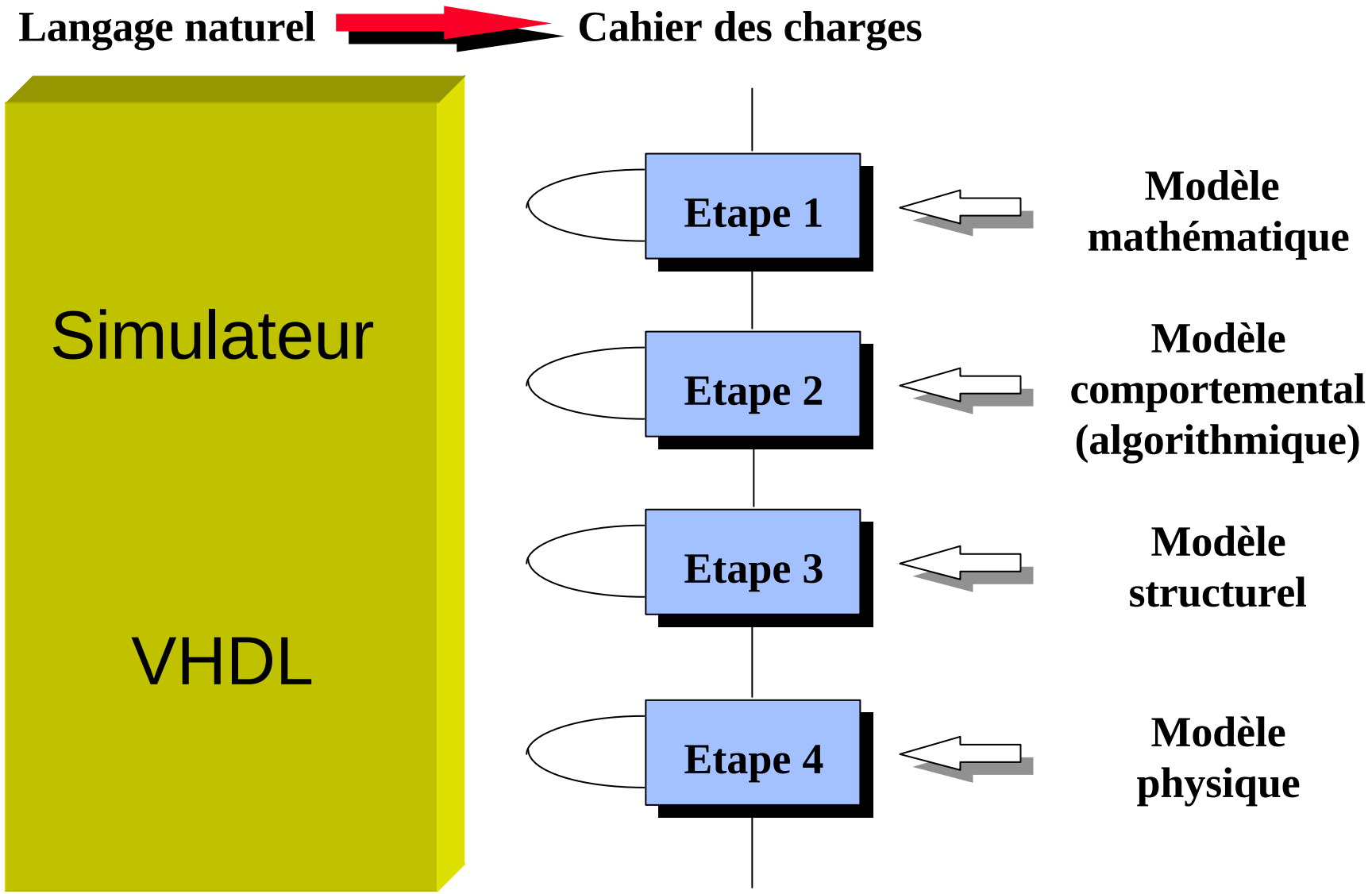


  - Un seul langage pour tous les niveaux de conception (VHDL)
  - Echange entre concepteurs

2) Méthodes de conception



2) Méthodes de conception



2) Méthodes de conception

- Modèles pour la conception : Synthèse de systèmes numériques

Niveaux	Primitives	Représentations
Système	fonctions	Spécif de performances
Module hard	micro, port, ram	Réponse E/S, algorithme
Machine séquentielle	Reg, UAL, Bus, mux	Table de vérité, diag d'états, programme
Logique	Portes	Equations booléennes
Circuit	Transistors, R, L, C	Equations différentielles
Silicium	Objets géométriques	

2) Méthodes de conception

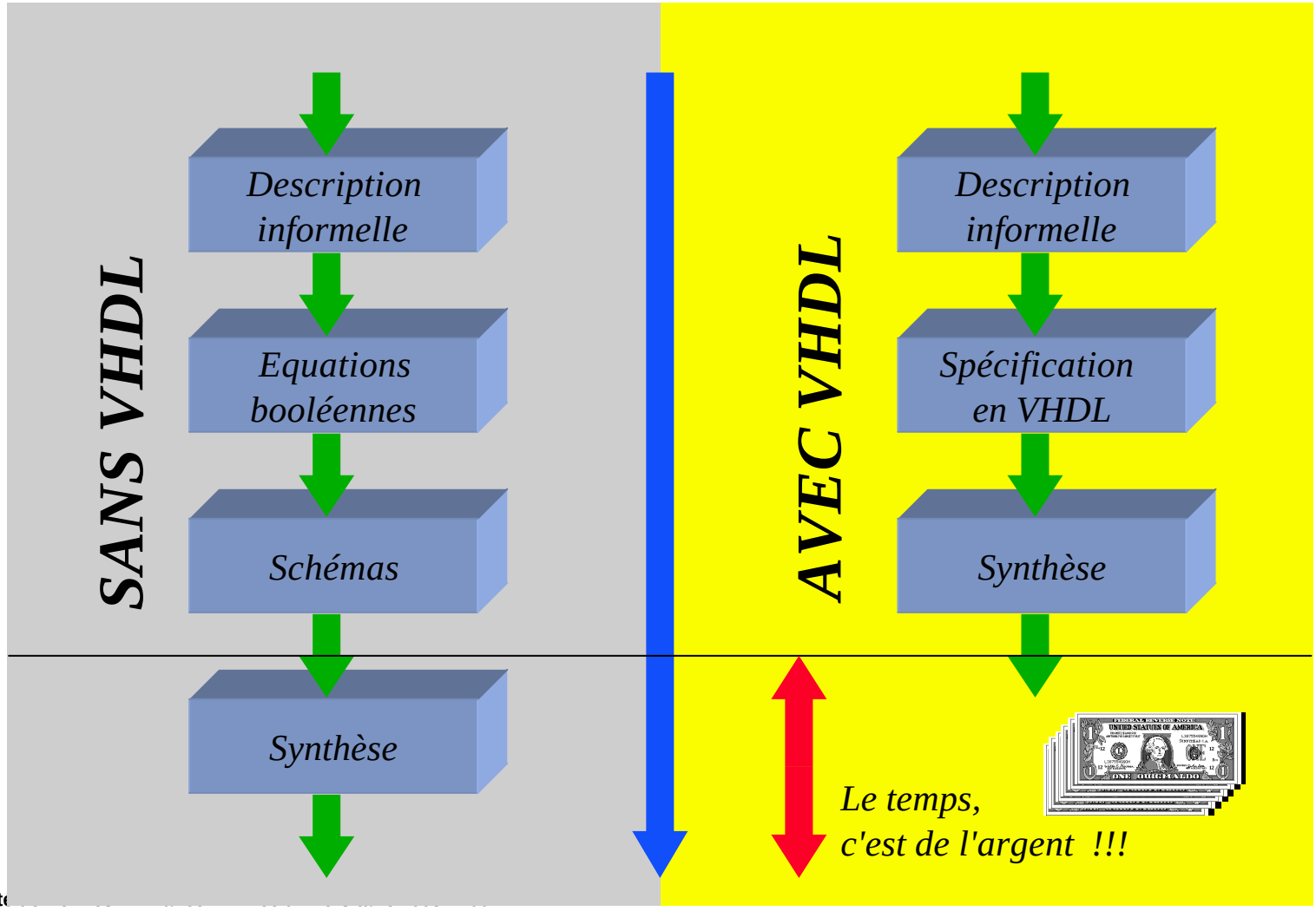
Modèles de description VHDL :

- Structurel
  - ✓ ne fait pas intervenir le temps
  - ✓ décrit la structure de la fonction réalisée
  - ✓ décrit un schéma, des connexions entre composants
- Comportemental
  - ✓ algorithme
  - ✓ le temps peut intervenir
- Flot de données
  - ✓ exprime le flot de données sortants par rapport au flot entrant

Possibilité de mélanger ces modèles de description

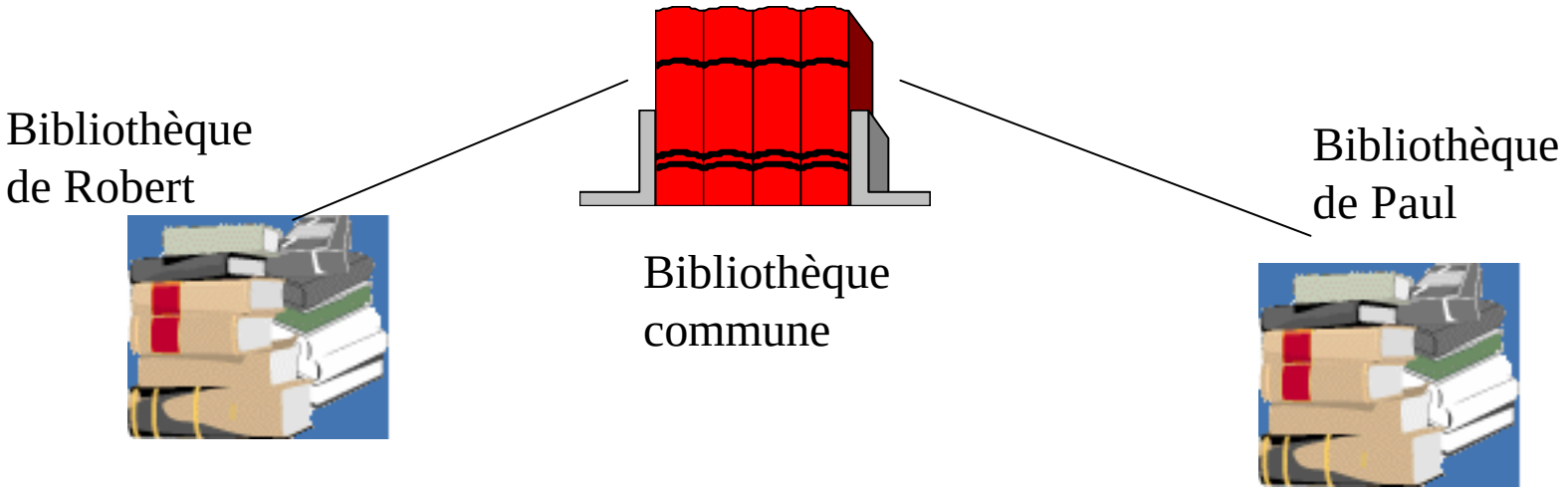
2) Méthodes de conception

- Intérêt de VHDL dans la conception descendante



2) Méthodes de conception

- Organisation de la conception :
  - Découpage en unités de conception :
    - ✓ ces unités s'auto suffisent
    - ✓ elles peuvent être compilées séparément
    - ✓ si la compilation est correcte alors les unités sont placées dans la bibliothèque de travail
  - Possibilité de partager les bibliothèques de ressources



2) Méthodes de conception

Bibliothèques et librairies

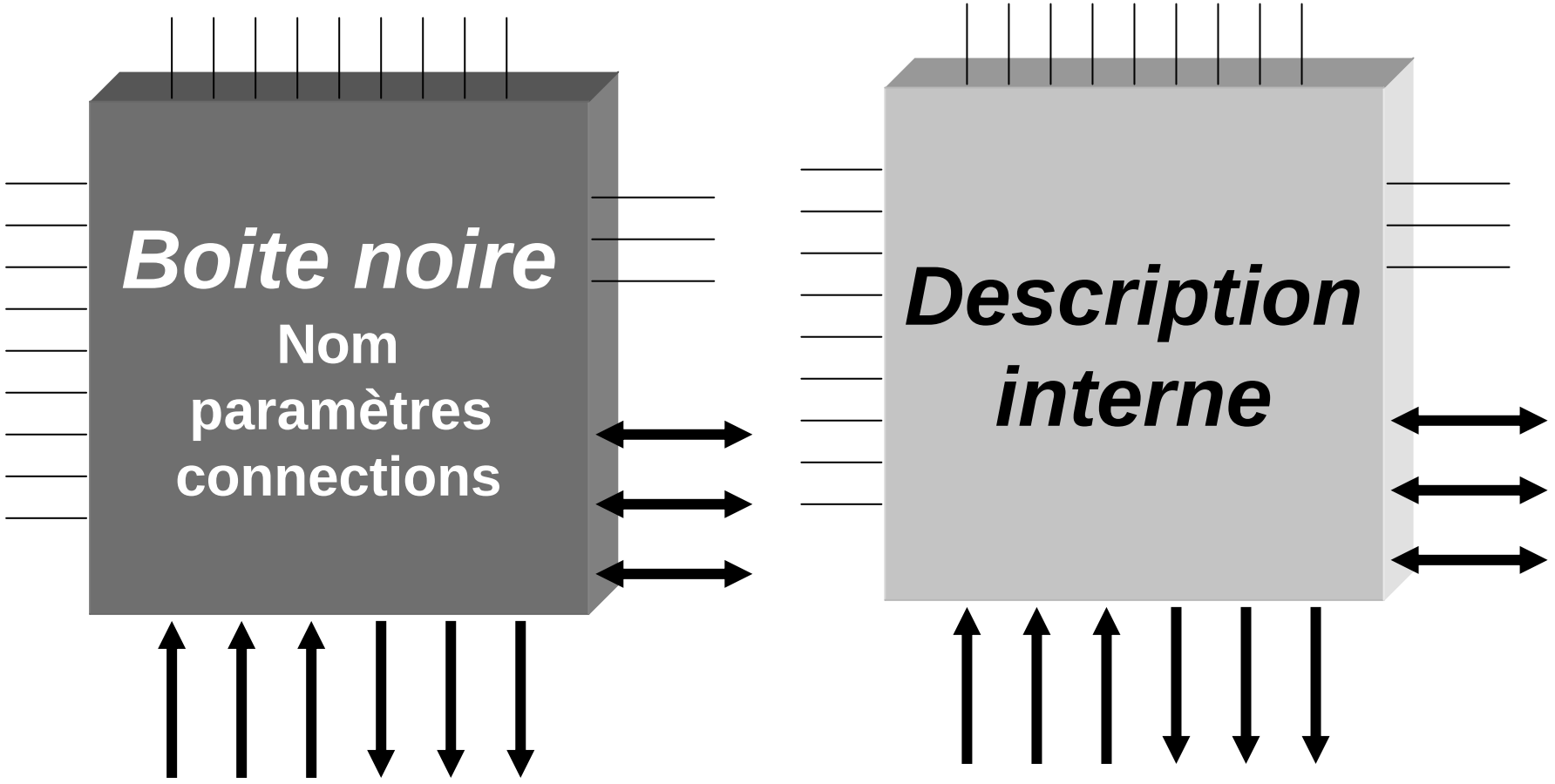
- A un instant donné, on travail avec la bibliothèque de travail :
  - ✓ choix de cette bibliothèque :
    - fait en dehors du langage VHDL
    - par une commande liée au système de développement
- Il est possible d'utiliser d'autres bibliothèques qui sont alors des bibliothèques de ressources :
- Par défaut, les bibliothèques WORK et STD sont inclues :
  - ✓ elles permettent la manipulation des "objets" standards :
    - bit
    - vector de bit
    - string
    - integer
    - time
    - etc

Unités de conception VHDL

3) Unités de conception

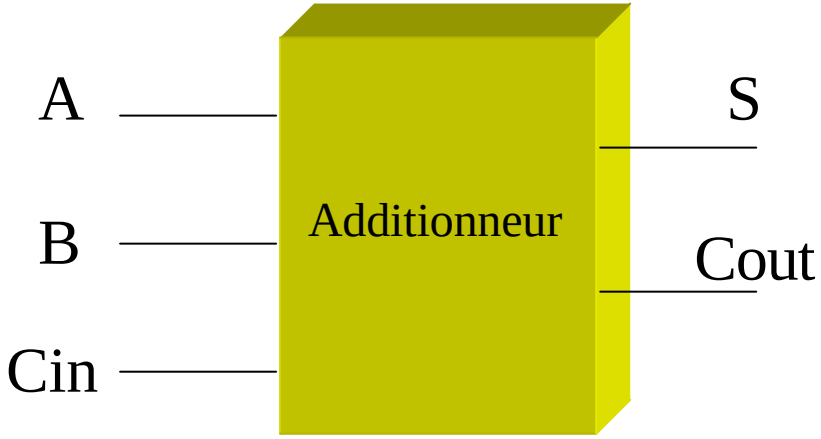
- Elles sont toujours constituées de 2 parties :

vue externe **vue interne**

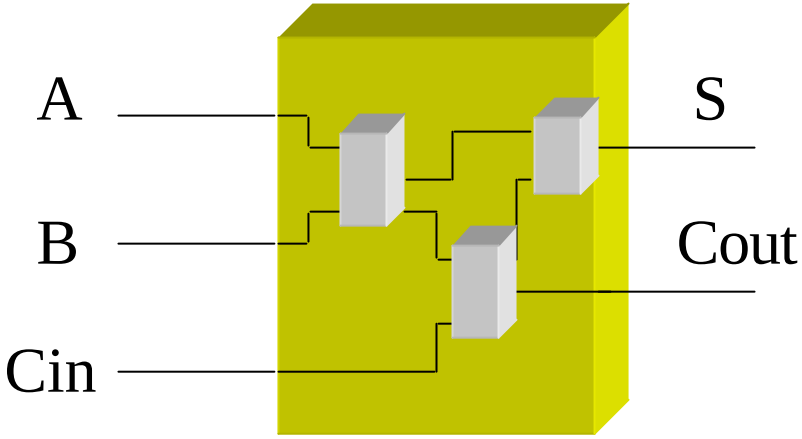


3) Unités de conception

- Exemples :
  - composants **vue externe**



vue interne



  - packages **vue externe**

déclaration de constantes
déclaration de types
déclaration des entêtes des fonctions

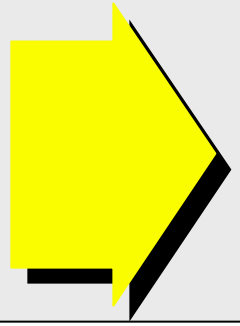
vue interne

description du corps de toutes les fonctions (y compris les fonctions non visible de l'extérieur)

3) Unités de conception

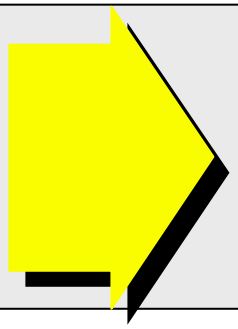
- 5 unités de conception disponible en VHDL :

  - spécification d'entités
    - ✓ (vue externe de la boite)
  - architecture
    - ✓ (intérieur de la boite)

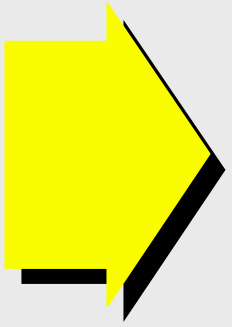
  Composant

  - configuration
    - ✓ (correspondance composants entités)

  Liens entre les entités et les composants utilisés

  - spécification de paquetage
    - ✓ (décrits les types, et sous programmes)
  - corps de paquetage
    - ✓ (décrits le corps des sous programmes)

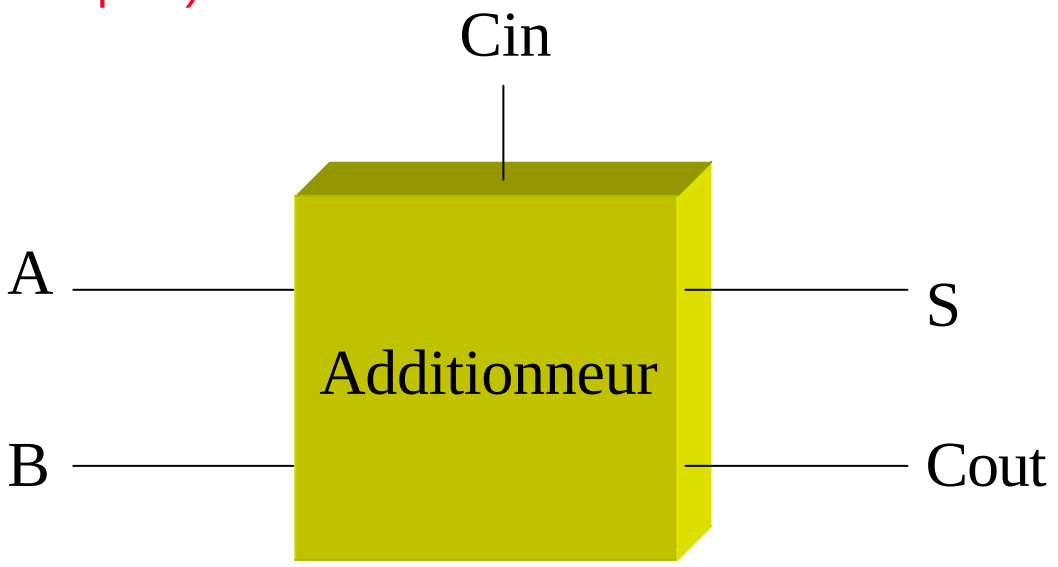
  

  Paquetage

3) Unités de conception

- Spécification d'entité :
  - Définit la vue externe :
    - ✓ nom
    - ✓ entrées sorties :
      - nombre
      - sens
      - type de données transportées
    - ✓ (paramètres génériques)

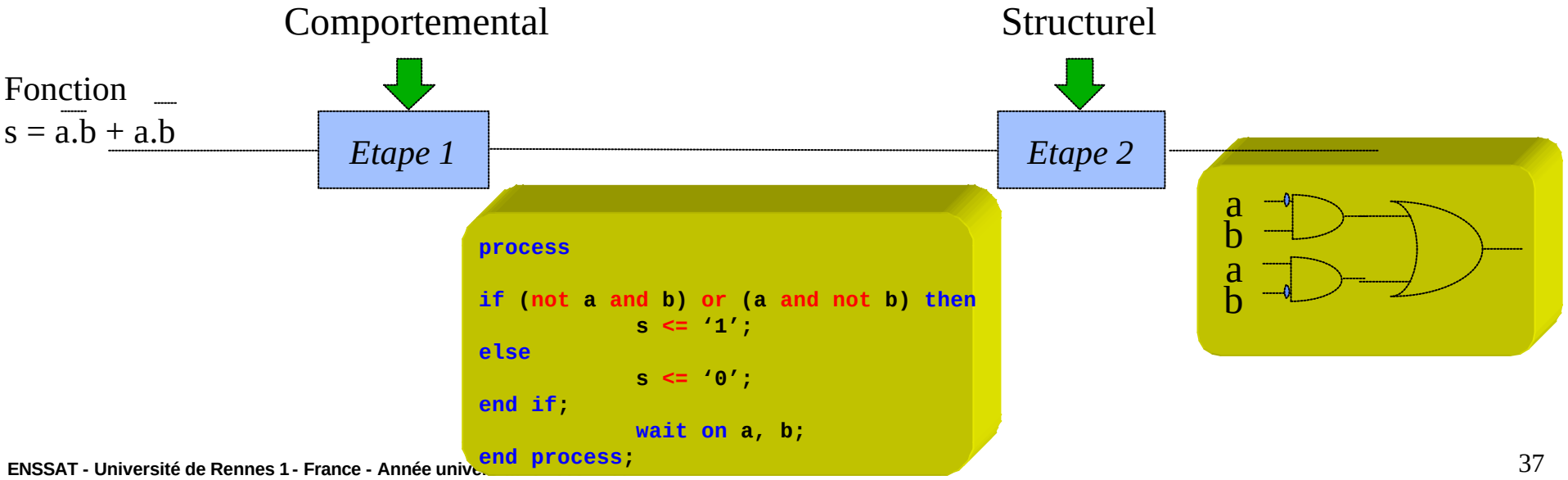
```
entity Additionneur is
        port ( A, B, Cin : in bit;
                  S, Cout : out bit)
end Additionneur ;
```



3) Unités de conception

Spécification d'architecture :

- Définit ce que fait l'entité
- 3 modèles sont utilisables :
  - ✓ comportemental — ***ce que fait l'entité***
  - ✓ structurel — ***comment elle le fait***
  - ✓ flot de données
- Plusieurs architectures peuvent être définies pour une entité (représentation à des niveaux d'abstraction différents)



3) Unités de conception

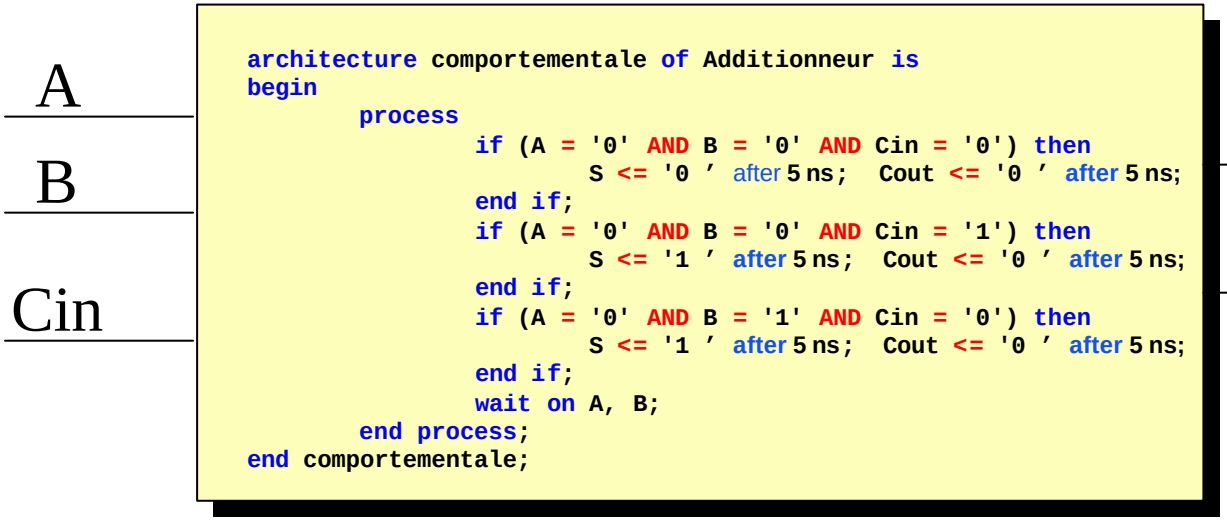
- Modèle comportemental :
  - description du fonctionnement du système :
    - ✓ algorithmes, machine d'états
  - algorithme proche de ceux des langages de programmation
  - le temps peut intervenir :
    - ✓ permet de coller à la réalité par le respect des temps de traversée
  - une description haut niveau est d'abord comportementale :
    - ✓ elle sert de référence pour la suite de la conception

3) Unités de conception

Modèle comportemental

- algorithmes, équations logique, ...

Additionneur




```
architecture comportementale of Additionneur is
begin
        process
                if (A = '0' AND B = '0' AND Cin = '0') then
                        S <= '0 ' after 5 ns;  Cout <= '0 ' after 5 ns;
                end if;
                if (A = '0' AND B = '0' AND Cin = '1') then
                        S <= '1 ' after 5 ns;  Cout <= '0 ' after 5 ns;
                end if;
                if (A = '0' AND B = '1' AND Cin = '0') then
                        S <= '1 ' after 5 ns;  Cout <= '0 ' after 5 ns;
                end if;
                wait on A, B;
        end process;
end comportementale;
```

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3) Unités de conception

Modèle comportemental

algorithme

```
architecture comportementale2 of Additionneur is
begin
        process (A, B, Cin)
        begin
                if (A = '0' AND B = '0' AND Cin = '0') then
                        S <= '0 ' after 5 ns;  Cout <= '0 ' after 5 ns;
                end if;
                if (A = '0' AND B = '0' AND Cin = '1') then
                        S <= '1 ' after 5 ns;  Cout <= '0 ' after 5 ns;
                end if;
                if (A = '0' AND B = '1' AND Cin = '0') then
                        S <= '1 ' after 5 ns;  Cout <= '0 ' after 5 ns;
                end if;
        end process;
end comportementale2 ;
```

A

B

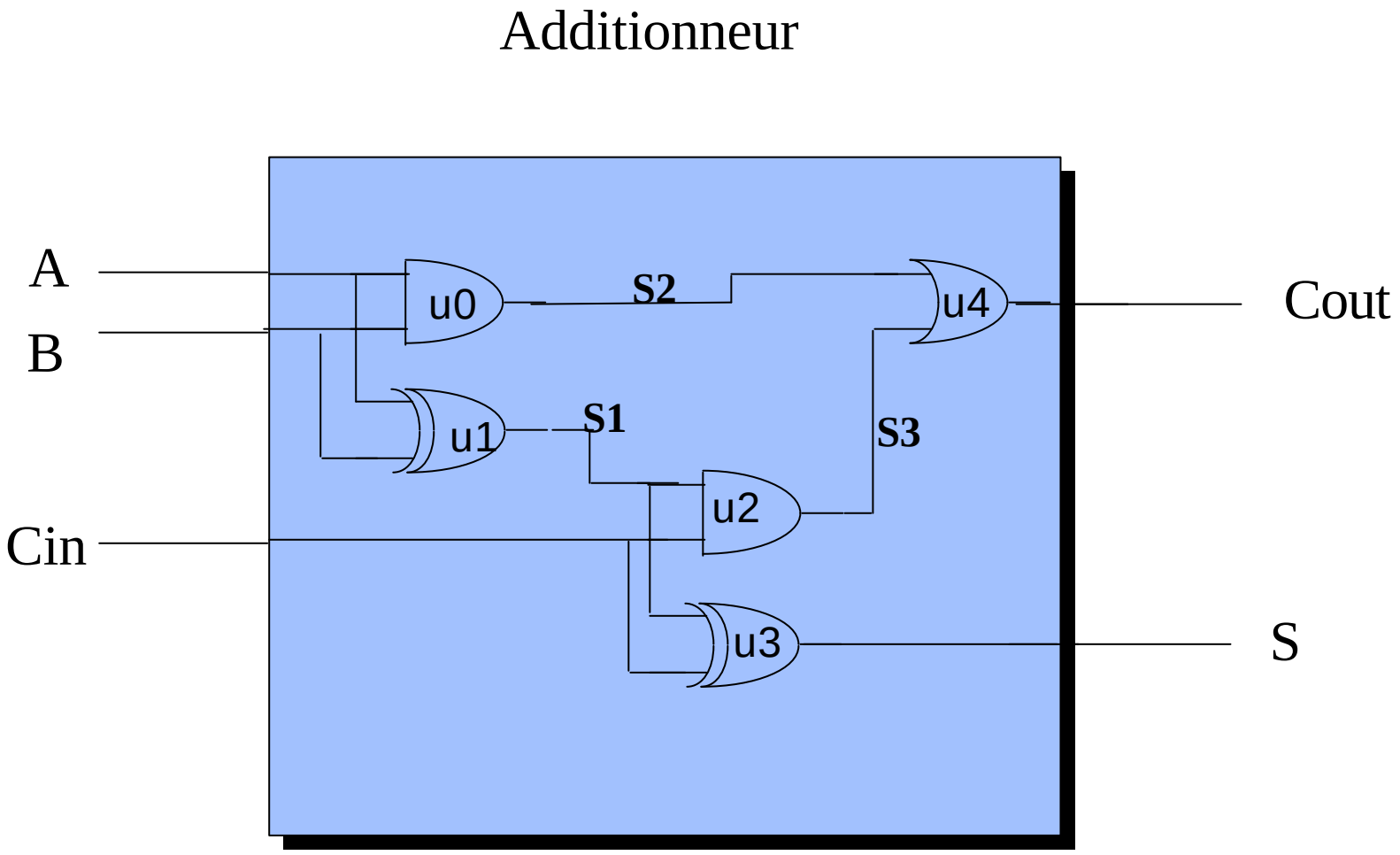
Cin

3) Unités de conception

- Modèle structurel
  - description par la structure :
    - ✓ liste d'éléments
    - ✓ interconnexions
  - ne fait pas intervenir le temps :
    - ✓ connexions permanentes
  - une feuille au moment de la simulation ne peut être décrite structurellement

3) Unités de conception

- Modèle structurel



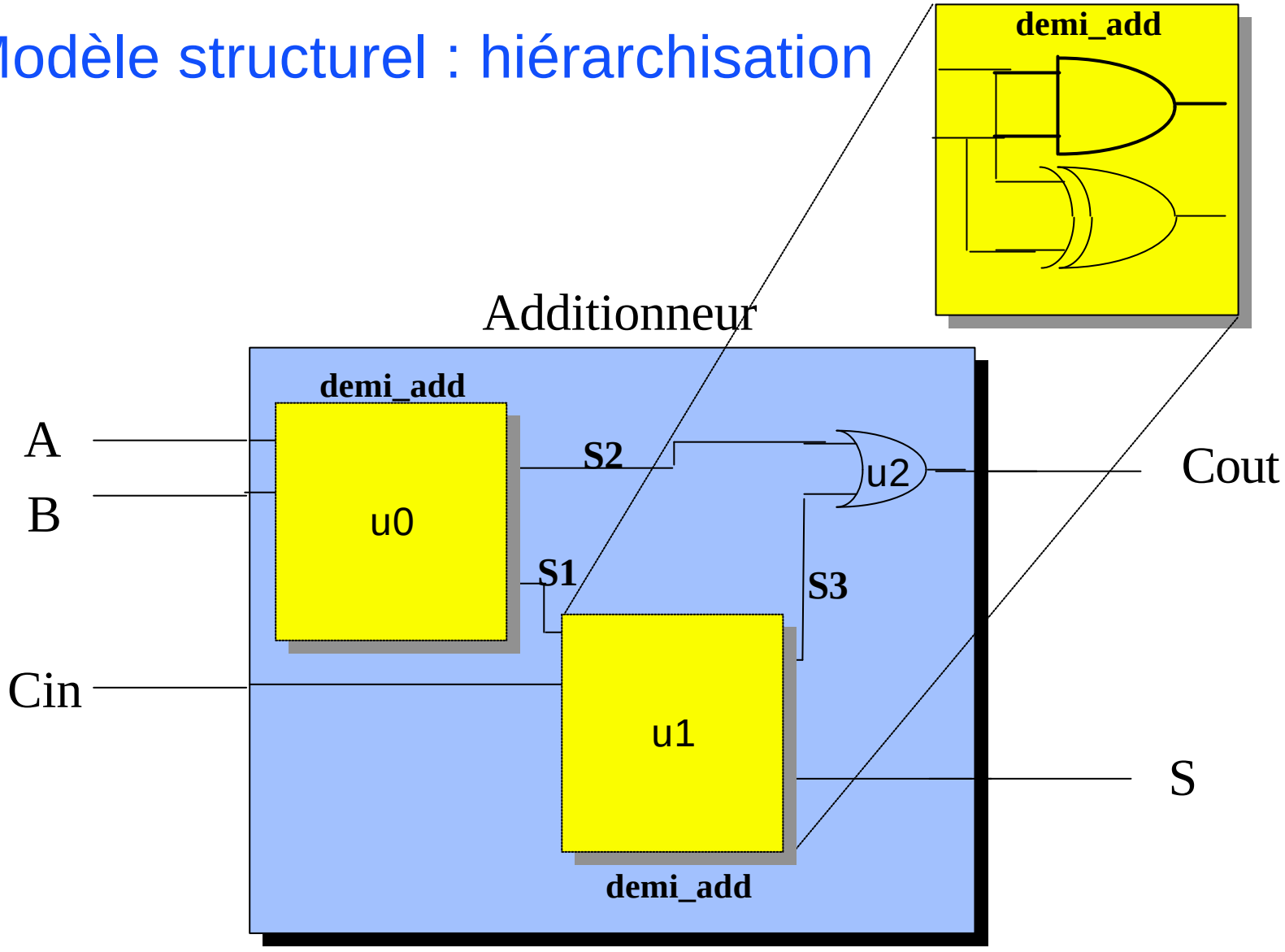
3) Unités de conception

```
architecture  structurelle1 of Additionneur is

    component porteOU
          port (    e1 : in bit;
                    e2 : in bit;
                    s : out bit );
    end component;
    component porteET
          port (    e1 : in bit;
                    e2 : in bit;
                    s : out bit );
    end component;
    component porteXOR
          port (    e1 : in bit;
                    e2 : in bit;
                    s : out bit );
    end component;

    signal S1, S2, S3 : bit;
```

```
begin
          u0 : porteET
                 port map ( A, B, S2);

          u1 : porteXOR
                 port map ( A, B, S1);

          u2 : porteET
                 port map ( S1, Cin, S3);

          u3 : porteXOR
                 port map ( S1, Cin, S);

          u4 : porteOU
                 port map ( S2, S1, Cout);

end structurelle1;
```

3) Unités de conception

- Modèle structurel : hiérarchisation



3) Unités de conception

```
architecture  structurelle2 of Additionneur is

          component porteOU
                    port (    e1 : in bit;
                              e2 : in bit;
                              s : out bit );
          end component;
          component DemiADD
                    port (    e1 : in bit;
                              e2 : in bit;
                              s1 : out bit ;
                              s2 : out bit);
          end component;

          signal S1, S2, S3 : bit;

begin
          u0 : DemiADD
                 port map ( A, B, S2, S1);

          u1 : DemiADD
                 port map ( S1, Cin, S3, S);

          u2 : porteOU
                 port map ( S2, S3, Cout);

end structurelle2;
```

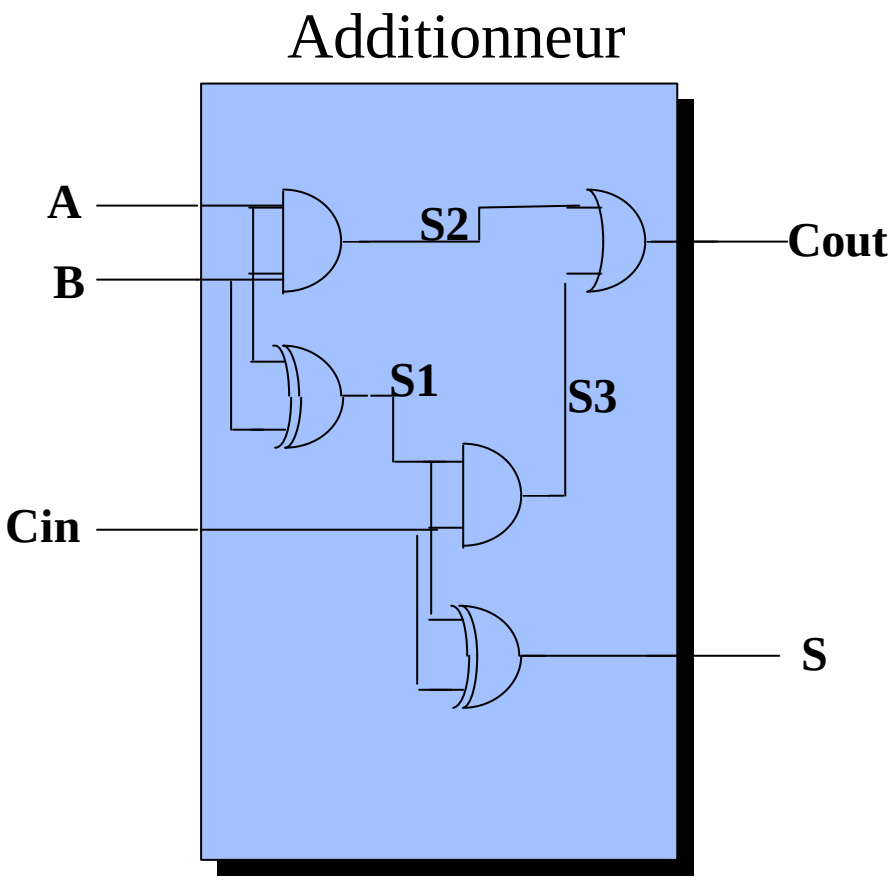
3) Unités de conception

- Avec DemiAdd :

```
Entity DemiAdd is
        port (
                e1 : in bit;
                e2 : in bit;
                s1 : out bit ;
                s2 : out bit);
end DemiAdd;

architecture  structurelle of DemiAdd is

        component porteET
                port (    e1 : in bit;
                          e2 : in bit;
                          s : out bit );
        end component;

        component porteXOR
                port (    e1 : in bit;
                          e2 : in bit;
                          s : out bit );
        end component;

begin
        u0 : PorteET
               port map ( e1, e2, s1);

        u1 : porteXOR
               port map ( e1, e2, s2);

end structurelle;
```

3) Unités de conception

Modèle flot de données




```
architecture flot of Additionneur is
          signal S1, S2, S3 : bit ;
begin
          S1 <= A xor B after 10 ns ;
          S2 <= A and B after 5 ns ;
          S3 <= S1 and Cin after 5 ns ;
          S   <= S1 xor Cin after 10 ns ;
          Cout <= S2 or S3 after 5 ns ;
end flot ;
```

```
architecture flot1 of Additionneur is
          signal S1, S2, S3 : bit ;
begin
          S   <= S1 xor Cin after 10 ns ;
          Cout <= S2 or S3 after 5 ns ;
          S1 <= A xor B after 10 ns ;
          S2 <= A and B after 5 ns ;
          S3 <= S1 and Cin after 5 ns ;
end flot ;
```

L'ordre des descriptions n'a pas d'importance !!!

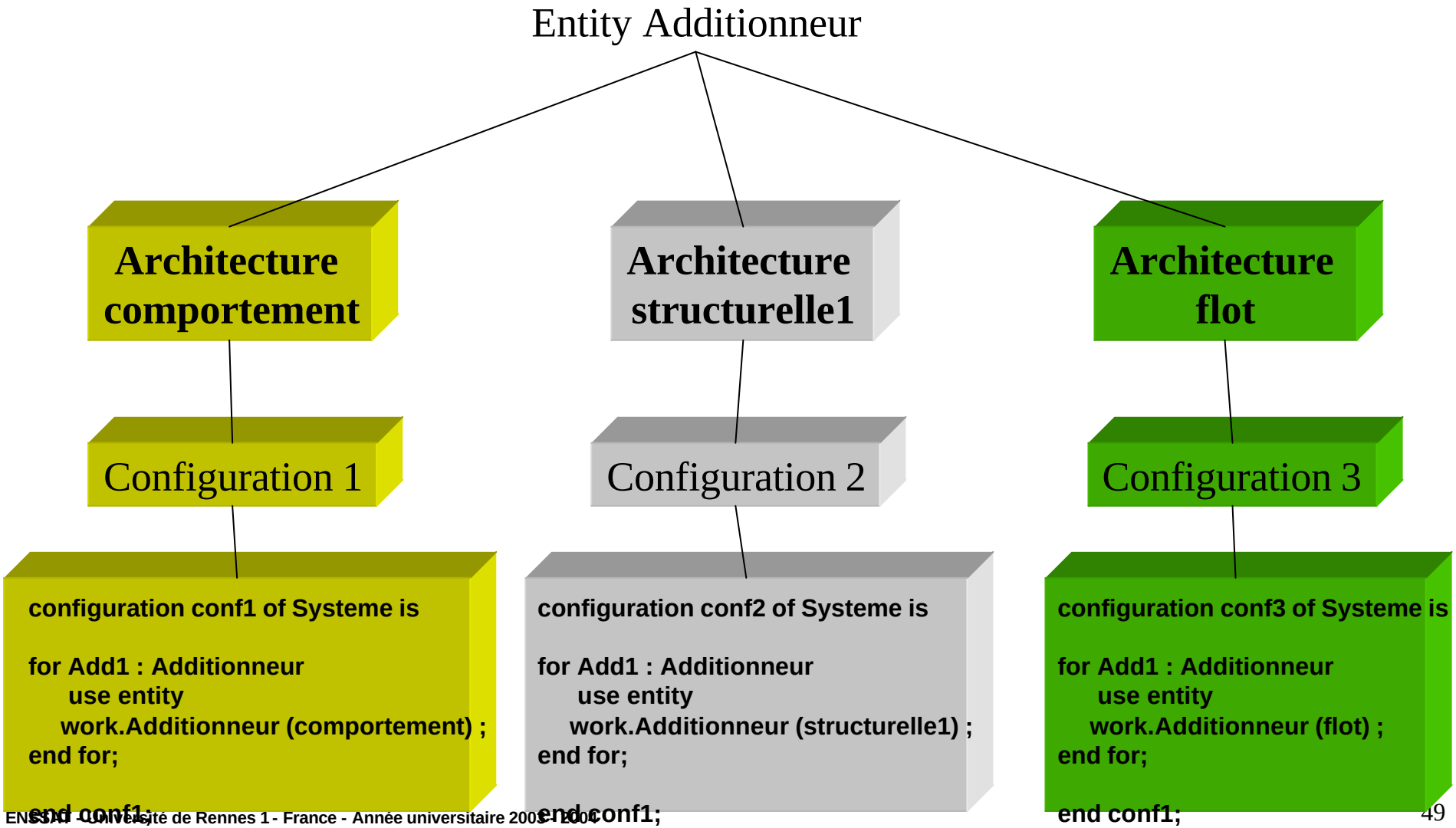
3) Unités de conception

- Spécification de configuration :
  - Effectue la liaison entre les composants utilisés dans une architecture et leur réalisation effective
  - Chaque utilisation d'un composant doit donner lieu à une configuration, c.a.d qu'il faut spécifier le modèle dont le composant est l'instance
  - Indique pour chaque composant, le couple entité / architecture choisie pour la réalisation
  - Indique les correspondances entre les ports du composant et de son modèle

Pour les instances label1, label2, etc du composant comp, on doit utiliser l'entité ent associé à l'architecture arch en prenant soin de faire correspondre tels ports aux ports formels de la même entité

3) Unités de conception

Soit un système utilisant un composant Add1 :



3) Unités de conception

- Spécification de package :

```
package nom   is

        constant PI : real ;      -- constante a valeur differee

        type boolean is (FALSE, TRUE);

        type couleur is (bleu, rouge, vert);

        type caracteres is (‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ...., ‘9’, ‘0’,
                            ‘a’, ‘b’, ...’z’ );

        subtype positif is integer range 0 to integer’high;

        procedure min (a: in integer ; b : in integer ; c : out integer);

        function max (a : in integer ; b : in integer )  return integer;

        .....

end nom;
```

3) Unités de conception

- Corps du package :

```
package body nom  is

        constant PI : real := 3.1415 ;

        procedure min (a: in integer ; b : in integer ; c : out integer) is
                variable ....
                begin
                    ....
        end min;

        function max (a : in integer ; b : in integer )  return integer is
                variable ....
                begin
                ....
                return maximum ;
        end max;

end nom ;
```

Les objets

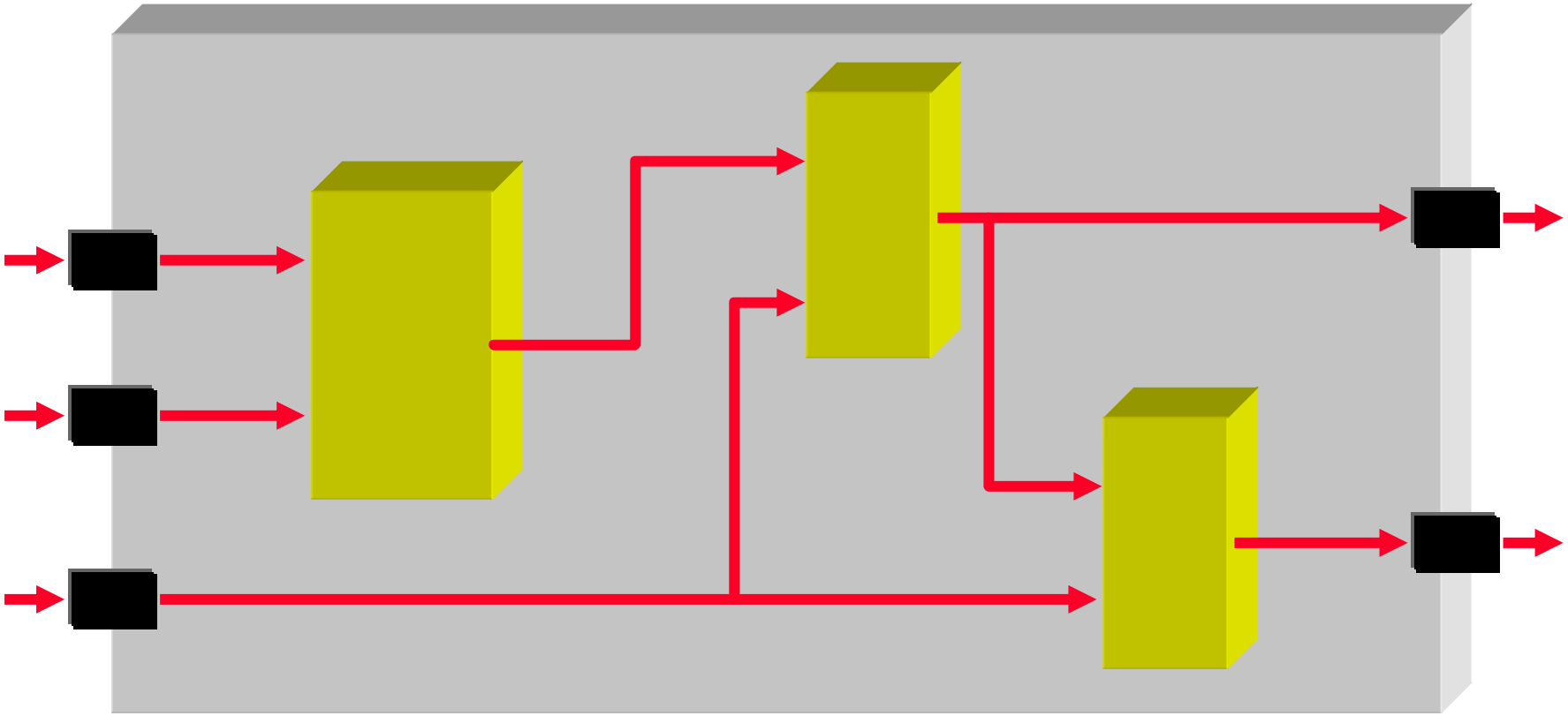
4) Les objets

- Constantes :
  - valeur fixe après initialisation
- Variables :
  - valeur modifiable par affectation (affectation instantanée)
- Signaux :
  - spécifiques aux langages de description de matériel
  - modélisent les informations qui circulent sur les fils ou bus
  - gérés par un pilote (driver)

  - **S <= '1', '0' after 10 ns, '1' after 20 ns, '0' after 30 ns, …**

4) Les objets

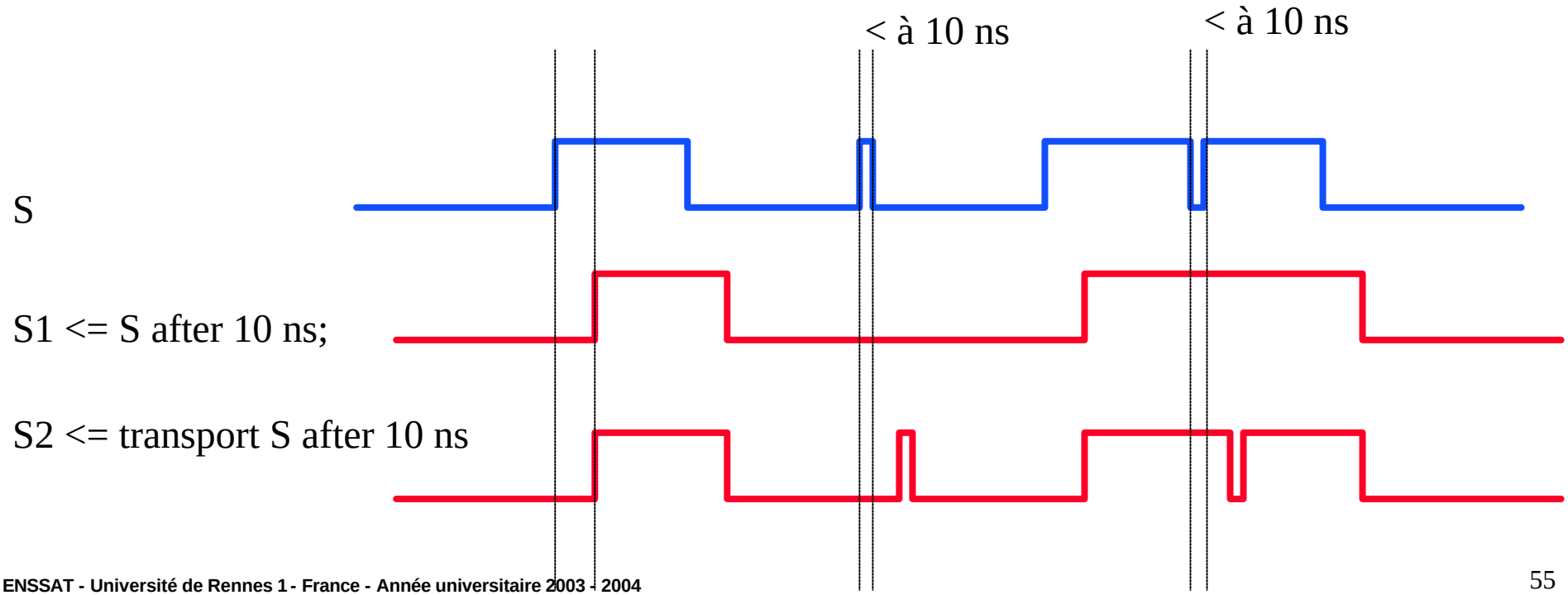
- Les signaux :
  - ils sont la base des langages de description de matériel
  - ils véhiculent les informations entre composants
  - ils ne sont pas de type pointeur, pas de type fichier



4) Les objets

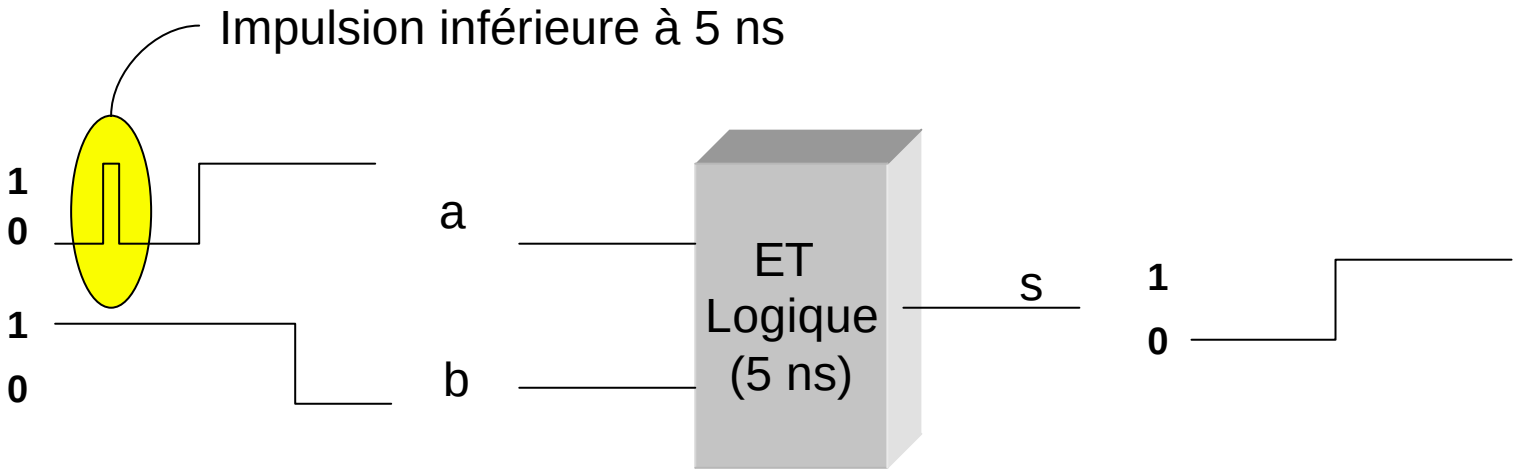
Affectation de signaux :

- 2 modes de fonctionnement :
  - ✓ mode inertiel :
    - filtre les impulsions de longueur insuffisante
  - ✓ mode transmission :
    - transmission de toutes les impulsions



4) Les objets

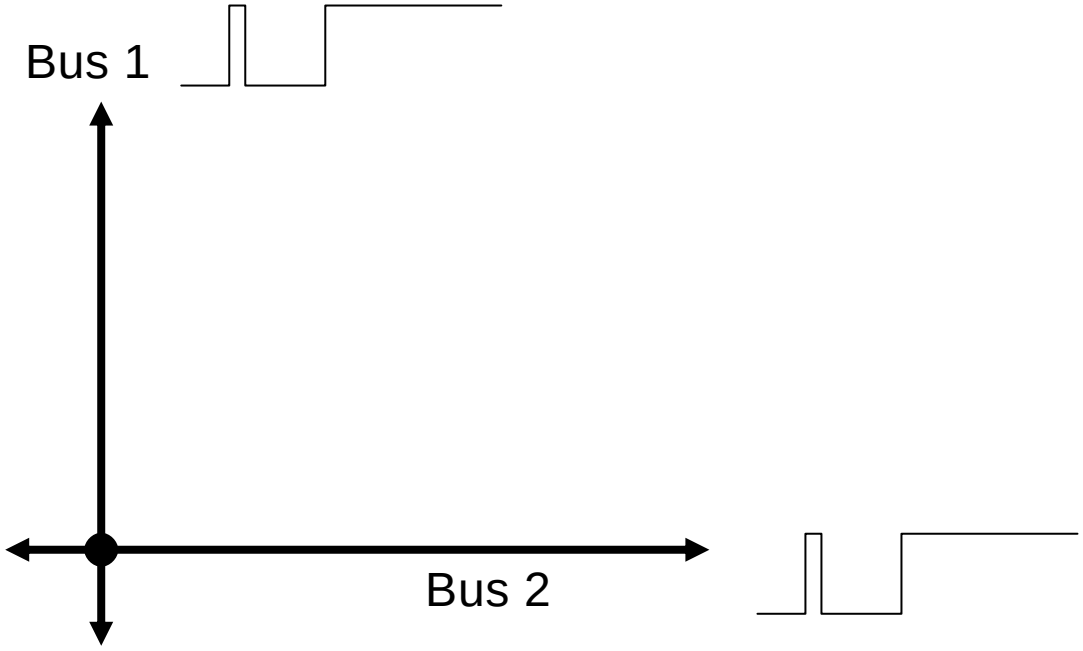
- Quel mode utiliser et quand ?
  - ✓ Mode inertiel
    - pour les composants dont le temps de traversé est non nul (tous les composants décrits de façon réaliste)




```
S <= A and B after 5 ns ;
```

4) Les objets

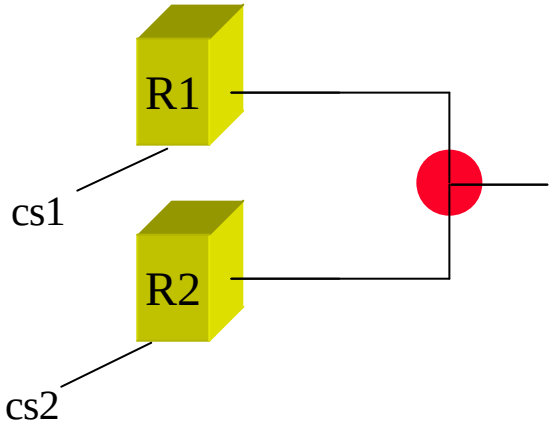
- ✓ Mode transport




```
Bus1 <= transport Bus2 ;
```

4) Les objets

- Affectations concurrentes de signaux :
  - problème matériel sous jacent :
    - ✓ le contrôleur doit s'assurer que les deux sorties de registres ne sont pas à l'état basse impédance simultanément



  - VHDL permet la constatation d'un problème de ce type :
    - ✓ par la fonction de résolution
    - ✓ la résolution du conflit sera réalisée durant la simulation du système

4) Les objets

Les types

- Scalaires :
  - entiers — **integer**
  - flottants — **real**
  - types physiques — **time**
  - énumérés — **(bleu, rouge, jaune, ...)**
- Composites :
  - tableaux — **array**
  - articles — **record**
- Accès:
  - pointeurs — **access (new, deallocate)**
- Fichiers (accès séquentiel, typé — **file (read, write, endfile)**

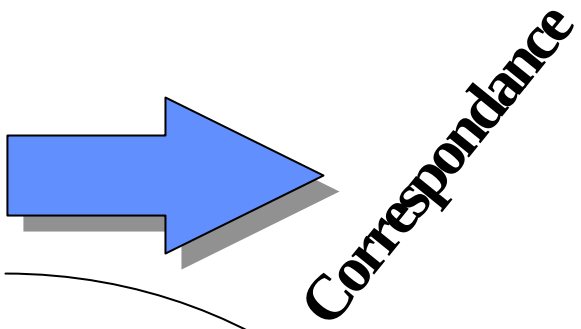
4) Les objets

Le type scalaire :

- il dispose d'une relation d'ordre classique
  - on peut les comparer (=, /=, <, >, <= et >=)
- restriction des valeurs possibles :
  - **range** valeur1 **to** valeur2
  - **range** valeur3 **downto** valeur4 (respect des habitudes des électroniciens, [poids fort ……. poids faible])
  - **subtype** positif **is integer range** 0 **to integer'high ;**
- type énuméré :
  - on indique les valeurs symboliques qui peuvent être prise
  - type couleur is (rouge, vert, bleu);
  - type etat is (marche, arret, attente);
  - le type caractère est un type énuméré,
- type physique :
  - le type time est prédéfini dans le paquetage standard

4) Les objets

- un type physique est caractérisé par :
  - ✓ son ***unité***
  - ✓ par des sous unités
  - ✓ on peut effectuer des calculs sur ces types




```
type TIME is range -LimiteInferieure to LimiteSuperieure
        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;
```

36 fs + (20 ps / 2) + 2 ns

4) Les objets

Les types composites :

- Tableaux et articles (array & record) :
  - ✓ tableau : éléments homogènes, de même type
  - ✓ articles : éléments hétérogènes, types différents
- Les tableaux :
  - ✓ dimensions multiples
  - ✓ les indices sont de type discret (énuméré ou entier)
  - ✓ 2 sortes de tableaux :
    - contraint : on spécifie le sens de variation des indices

1 to 15 **15 downto 1**

```
type mot is array  (15 downto 0)  of bit;
type memoire is array  (0 to 255)  of mot;
```

4) Les objets

- Les tableaux non contraint :
  - ✓ permet de repousser la définition de la taille à plus tard.
  - ✓ la manipulation de ces tableaux demande l'utilisation d'attributs

range <>

```
type string is array (positive range <>) of character;
type bus is array (positive range <>) of bit;
```

4) Les objets

Les articles :

- des éléments de type différents
- les champs sont désignés par un nom
- les éléments sont désignés par la notation pointée

```
type article is record
      champs1 : integer ;
      champs2 : bit ;
end record;
```

- affectation par champs ou globale

```
A.champs1 := B.champs1 ;
A := B ;
```

4) Les objets

Notation par agrégat

- indication de la valeur d'une variable de type composite

```
type tableau is array (0 to 4) of integer;

type article is record
        champs1 : integer;
        champs2 : bit;
        champs3 : integer;
end record;
```

- ✓ par défaut : association "positionnelle"

```
A: tableau := (5, 3, 1, 7, 9);
B : article := (12, '1', 9);
```

- ✓ association par nom :

```
A: tableau := (1=>3, 4=>9, 0=> 5, 2=>1, 3=>7 );
B : article := (champs2=>'1',champs3=> 9, champs1=> 12);
```

4) Les objets

- ✓ association mixte :
  - doit commencer par la partie "nominative"

```
A: tableau := (4=>9, 0=> 5, others => 0);
B : article := (champs2=>'1', others => 0);
```

équivalents à

```
A := (5, 0, 0, 0, 9);
B := (0, '1', 0);
```

- ✓ les champs affectés par **others** doivent être du même type
- ✓ affectation à zéro d'un bus de type non contraint

```
type bus is array (positive range <>) of bit;

...
signal BusData : bus (7 downto 0);
signal BusAddress : bus (15 downto 0);

....
BusData <= (Others => 'Z') ;
BusAddress <= (Others => '0') ;
```

Pas très générique !!!

```
BusData <= "ZZZZZZZZ";
BusAddress <= "000000....00000";
```

4) Les objets

Le type accès :

- Allocation dynamique de mémoire
- Pointeur sur un objet de type prédéfini
- Instructions :
  - ✓ new
  - ✓ deallocate

```
type article is record
      champs1 : integer;
      champs2 : bit;
end record;

type pointeur  is access article;

variable p, q : pointeur;

p:= new pointeur ;
q := new pointeur'(14, '0');          -- Initialisation
```

Peu utilisé !

```
deallocate(p);
```

4) Les objets

Le type fichier :

- utilisé pour :
  - ✓ fichiers de stimuli
  - ✓ charger le contenu d'une ROM (par exemple)
- Un fichier est typé (pas de pointeur, pas de tableaux à plus de 1 dimension, pas de type composite)
- 3 procédures sont créées implicitement :
  - ✓ lecture ; écriture ; fin de fichier :

```
type fich_txt is file of string;
type fich_int if file of integer;

txt : fich_txt ;    int : fich_int;
mot : string(1 to 10);  a : integer;

read (txt, mot);
write(int,a);
if endfile(txt) then
```

4) Les objets

- Les sous types :
  - compatible avec le type initial :
    - ✓ conserve les propriétés
    - ✓ restriction du type de base : restriction de l'espace des valeurs
    - ✓ subtype naturel is integer range 0 to integer'high;
    - ✓ subtype negatif is integer range integer'low to -1;
    - ✓ subtype index is integer range 2 to 5; -- indice d'un tableau
    - ✓ sous type dynamique :
      - subtype mot is VecteurDeBits (MAX-1 downto 0);
  - lisibilité des descriptions
  - sous types de sous types .....

Les éléments du langage

5) Les éléments du langage

- Opérations classiques :
  - **+, *, =, /=, ..., mod, and, or, …**
  - possibilité de surcharger les opérateurs, les fonctions, les procédures (de leur donner une autre signification)
- caractère : ‘X’
- chaînes de caractères : “.....”
- chaînes de bits :
  - **X“AB08”** : hexadécimale
  - **O“037”** : octale
  - **B“001101”** : binaire
- Commentaires : **-- ceci est un commentaire**

5) Les éléments du langage

- Initialisation des données, il s'agit d'une initialisation par défaut :
  - pour les types scalaires : il s'agit du premier élément de l'énumération (connue par l'attribut **left**)
  - pour les types composites : il s'agit des premiers éléments de chaque champs
  - pour les pointeurs : **null**

```
type couleur is (rouge, vert, bleu);  -- valeur par défaut est
rouge

subtype positif is integer range 0 to integer'high;
                                  -- valeur par défaut est 0
type  article is record
                        champs1 :  positif;
                        champs2 : couleur;
                        suivant : pointeur;
end record;                       -- valeur par défaut  ( 1, rouge, null)
```

5) Les éléments du langage

- Les sous programmes : procédures et fonctions
  - déclaration (optionnelle) : spécifie
    - ✓ le type (fonction ou procédure)
    - ✓ le nom
    - ✓ la liste des paramètres:
      - mode IN : pris par défaut, les données ne peuvent être que lues
      - mode OUT : ne peut être lu, inutilisable pour appeler un autre sous programme, inutilisable dans les fonctions (une fonction ne retourne que sa valeur)
      - mode INOUT : inutilisable dans les fonctions
    - ✓ le type de valeur de retour

```
Procedure MIN ( a, b : in integer; c : out integer)
```

```
Function MAX ( a, b : in integer) return  integer ;
```

5) Les éléments du langage

- Les sous programmes :
  - corps, contient l'algorithme
    - ✓ exemples

```
Procedure MIN ( a, b : in integer; c : out integer) is
begin
        if a<b then
                c := a ;
        else
                c:= b ;
        end if;
end MIN ;
```

```
Function MAX ( a, b : in integer) return  integer  is
begin
        if a<b then
                return  b ;
        else
                return  a ;
        end if;
end MAX ;
```

5) Les éléments du langage

- Les sous programmes :
  - appel de sous programme :
    - ✓ appel d'une procédure = instruction
    - ✓ appel d'une fonction = expression
    - ✓ passage de paramètres :
      - par position
      - par nom

```
MIN ( var, 5, resultat) ;

MIN ( b=> 5, a=> var, c=> resultat) ;
```

```
x := MAX ( var, 5) ;

x := MAX ( b=> 5, a=> var) ;
```

5) Les éléments du langage

- Surcharge d'opérateurs :
  - Addition de bit vector :
    - ✓ beaucoup d'outils dispose de bibliothèques permettant de réaliser cette opération
    - ✓ si elle n'est pas présente, on surcharge l'opérateur +

```
function "+" (A, B : in bit_vector ) return bit_vector;  -- opérateur binaire

function "+" (A : in bit_vector) return bit_vector ;      -- opérateur unaire

signal S1, S2,  S3, S : bit_vector ;
 ....

S <= S1 + S2 ;

ou

S <= "+" ( S1, S2 );          -- Appel classique des fonctions
S3 <= "+" (S1) ;
```

5) Les éléments du langage

Instructions séquentielles :

- wait :
  - ✓ suspend l'exécution d'un processus
  - ✓ plusieurs cas :
    - suspension jusqu'à la fin des temps : **wait**
    - suspension pendant un temps donnée **: wait for 10 ns;**
    - suspension jusqu'à événement sur signaux : **wait on S1, S2**
    - idem précédent plus condition : **wait on S1, S2 until condition**
    - idem précédent plus délai maximum d'attente :

      wait on S1, S2 until condition for 5 ms

  - ✓ interdit à l'intérieur d'une fonction (une fonction rend un résultat ***immédiat***, donc pas d'attente)

5) Les éléments du langage

- assert :
  - ✓ surveillance d'une condition
  - ✓ envoie de message sous condition

    assert (S1 = S2) report "Conflit de signaux" severity ERROR;

    si la condition est vrai, on ne fait rien sinon on affiche le message
  - ✓ 4 niveaux de sévérité :
    - NOTE, WARNING, ERROR, FAILURE
  - ✓ intéressant lors de la mise au point de code VHDL

Ne pas hésiter à l'utiliser et à placer des messages clairs, indiquant le nom de l'entité, de l'architecture, etc.

```
entity Additionneur is
                port (          A, B : in bit;
                                S : out bit)
end Additionneur;
architecture comportement of Additionneur is
begin
                process (A, B)
                begin
                                assert ((A AND B) = '0') report "Additionneur : Probleme de debordement" severity warning;
                                S <= A and B after 4 ns;
                end process;
end comportement;
```

5) Les éléments du langage

Affectation de signaux :

- modification des valeurs futurs que prendra le signal

```
S <= s1 after 10 ns, '0' after 50 ns, '1' after 100 ns;
a <= b after 3 ns;
c <= d ;          -- affectation avec délai delta
```

- les délais doivent être croissants
- notion de délai delta, il existe 2 dimensions du temps
  - ✓ temps "réel' : c'est le temps vu par le concepteur
  - ✓ temps "delta" : c'est le temps géré par le simulateur pour réaliser la succession des affectations

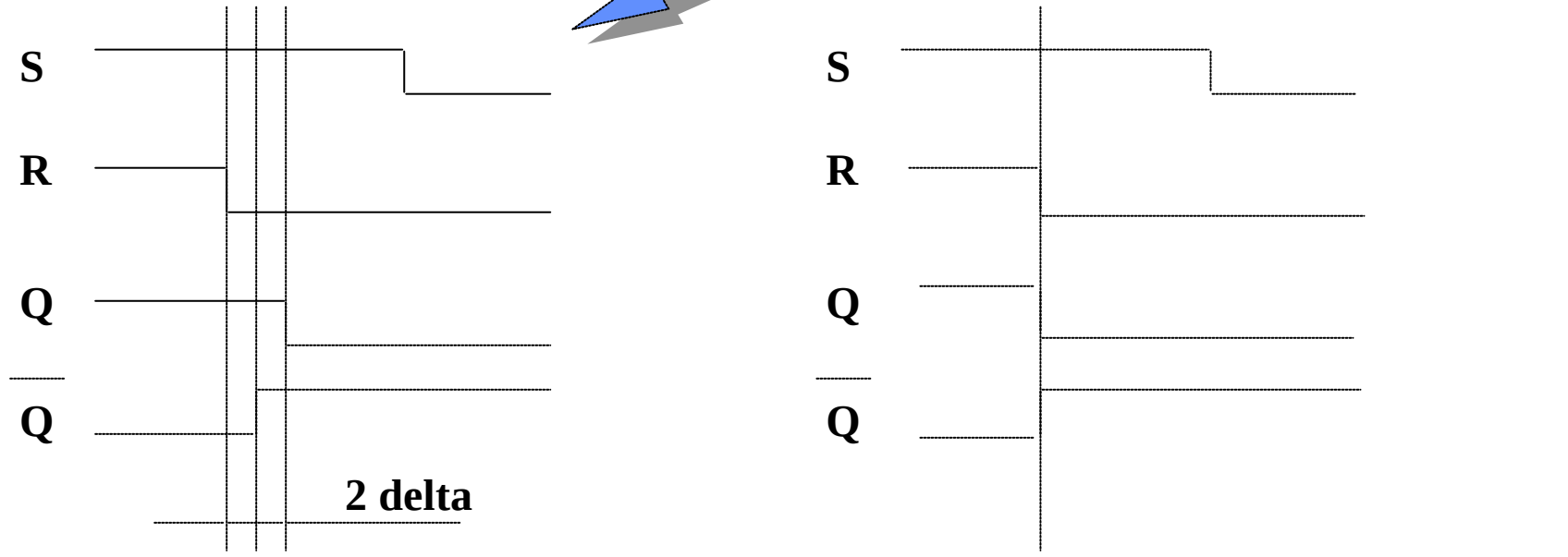
5) Les éléments du langage

- soient deux affectations :

```
Q <= S nand Q ;
Q <= R nand Q ;
```

Temps vu par :

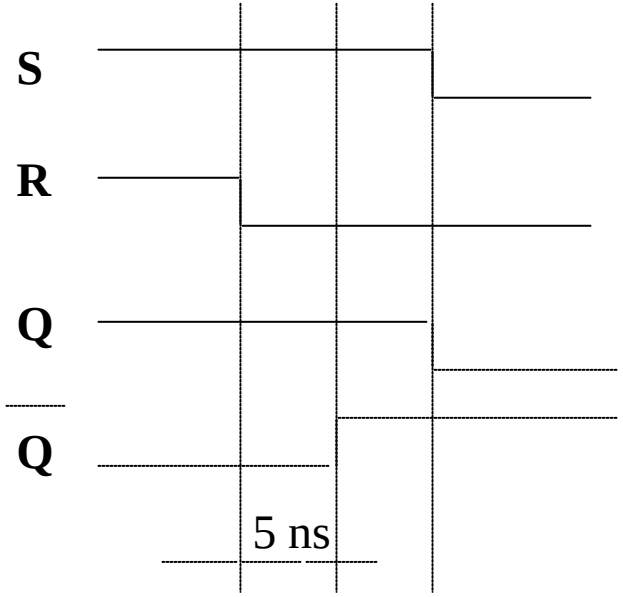
le simulateur **le concepteur**



5) Les éléments du langage

- Modélisation plus réaliste :
  - on tient compte des contraintes technologiques (contraintes matérielles)

```
Q <= S nand Q after 5 ns ;
Q <= R nand Q after 5 ns ;
```



5) Les éléments du langage

Instructions conditionnelles :

Structure *if then else endif* :

```
if condition_boolenne then
        sequence d'instructions 1 ;
else
        sequence d'instructions 2 ;
end if;
```

Structure *if then elsif else endif* :

```
if condition_1 then
        sequence d'instructions 1 ;
elseif condition_2 then
        sequence d'instructions 2 ;
elseif condition_3 then
        sequence d'instructions 2 ;
end if;
```

5) Les éléments du langage

- Structure case

```
case expression is
      when valeur1                => sequence d'instructions ;
      when valeur2 | valeur3      => sequence d'instructions ;
      when valeur4 to valeur5     => sequence d'instructions ;
      when others                 => sequence d'instructions ;
end case;
```

Très intéressant pour les descriptions de machine d'états

→ C'est une structure très facile à générer et à reconnaître

→ Les outils de synthèse (conception) reconnaissent très bien cette structure et savent en faire un schéma logique (transistors) optimisé

5) Les éléments du langage

Instructions de boucles :

- boucle infinie :

```
loop
      séquence d'instructions ;
end loop;
```

- boucle *while* :

```
while condition loop
      séquence d'instructions ;
end loop ;
```

- boucle *for* :

```
for i  in 1 to 10 loop
      séquence d'instructions ;
end loop;
```

La variable de boucle peut ne pas être déclarées

```
for i  in A'range loop
      séquence d'instructions ;
end loop;
```

5) Les éléments du langage

- Les attributs :
  - ils permettent de connaître les caractéristiques :
    - ✓ des signaux
    - ✓ des tableaux
    - ✓ des types
  - très utilisés pour rendre les descriptions génériques

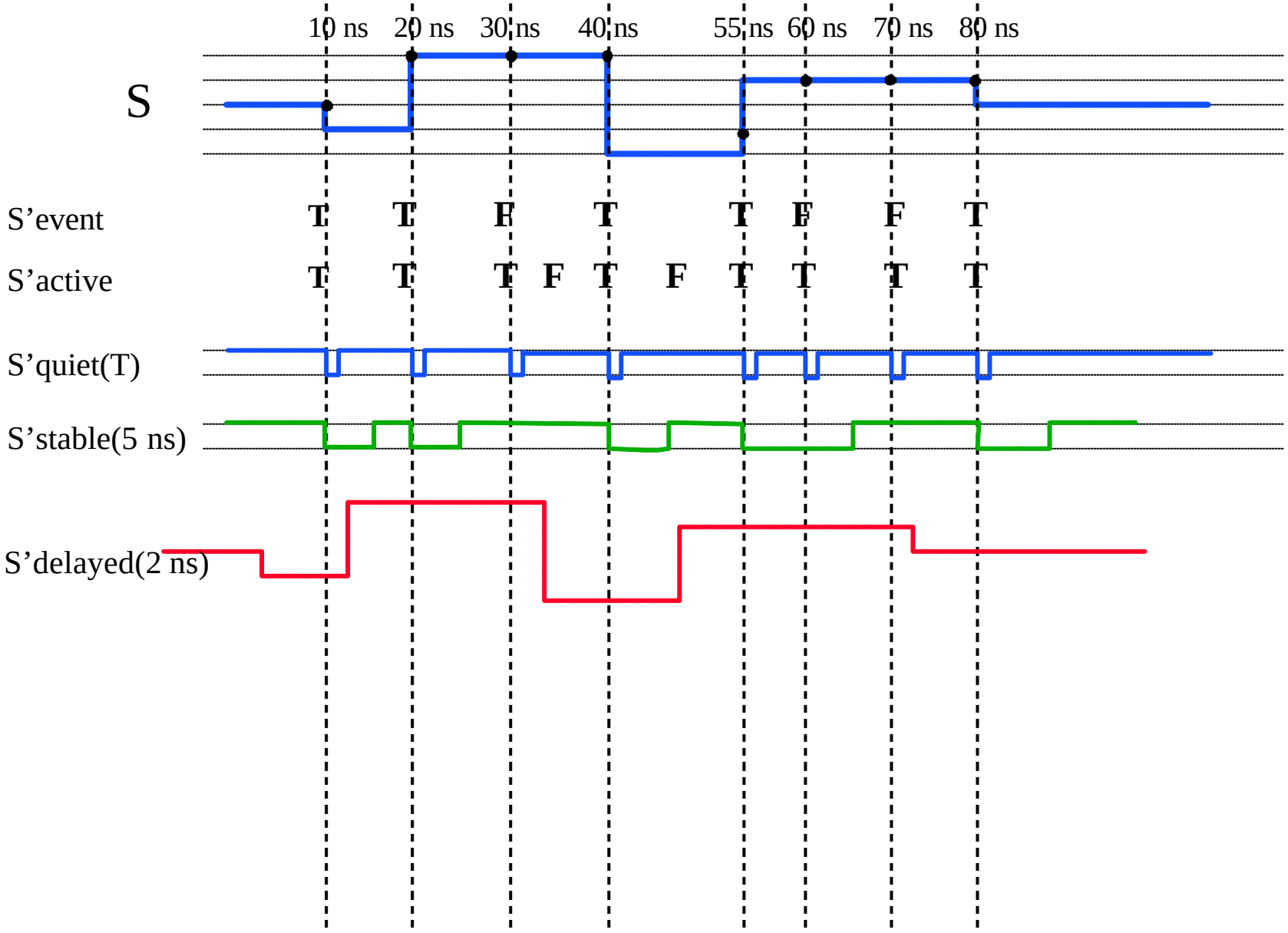
5) Les éléments du langage

- **Attributs de signaux :**
  - ✓ **S'event** :
    - boolean
    - true si un événement vient d'arriver sur S pendant le cycle de simulation en cours
  - ✓ **S'active** :
    - boolean
    - true si il y a eu une transaction (affectation) sur le signal dans le cycle de simulation en cours
  - ✓ **S'quiet(T)** :
    - boolean
    - true si le signal a eu ni transaction ni événement pendant un temps T
  - ✓ **S'stable(T)** :
    - boolean
    - true si il n'y a pas eu d'événement sur le signal pendant le temps T
  - ✓ **S'transaction** :
    - signal
    - c'est un signal de type bit qui change d'états pour chaque transaction du signal

5) Les éléments du langage

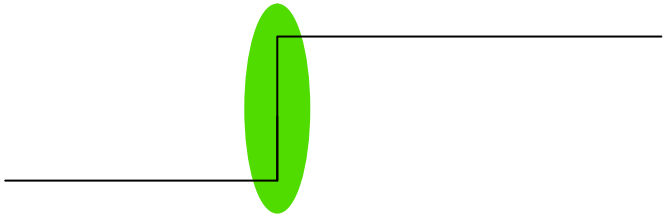
- ✓ **S'delayed** :
  - signal
  - c'est un signal identique à S mais retardé de T
- ✓ **S'last_event** :
  - time
  - rend le temps écoulé depuis le dernier événement sur le signal
- ✓ **S'last_active** :
  - time
  - rend le temps écoulé depuis la dernière transaction
- ✓ **S'last_value** :
  - type du signal
  - rend la valeur du signal immédiatement avant le dernier changement de S

5) Les éléments du langage



5) Les éléments du langage

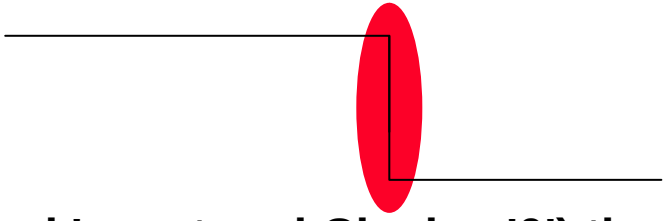
- Utilisation des attributs de signaux :
  - ✓ détection d'un front montant :



- **if (Clock'event and Clock = '1') then**

  - ✓ détection d'un front descendant :

Logique synchrone



- **if (Clock'event and Clock = '0') then**

5) Les éléments du langage

- Attributs sur les types
  - ✓ **type etat is (Arret, Marche, Monter, Descendre) ;**
  - ✓ **type NewBits ('0', '1', 'X', 'Z') ;**
  - ✓ **type BitBus is range 15 downto 0;**
  - ✓ **subtype positif is integer range 0 to integer'high;**

  - ✓ **positif'base :**
    - renvoie le type de base
      - dans notre cas renvoie integer
  - ✓ **BitBus'high ===> 15**
  - ✓ **BitBus'low ===> 0**
  - ✓ **BitBus'left ===> 15**
  - ✓ **NewBits'left ===> '0'**
  - ✓ **NewBits'right ===> 'Z'**
  - ✓ **etat'pos(Marche) ===> 1**
  - ✓ **etat'val(2) ===> Monter**
  - ✓ **etat'succ(Marche) ===> Monter**

5) Les éléments du langage

- ✓ **etat'pred(Marche) ===> Arret**
- ✓ **NewBits'leftof('Z') ===> 'X'**
- ✓ **NewBits'rightof('1') ===> 'X'**
- ✓ **BitBus'ascending ===> false**
- ✓ **positif'ascending ===> true**
- ✓ **etat'image(Marche) ===> "Marche"** chaîne de caractères
- ✓ **etat'value("Arret") ===> Arret** du type état

5) Les éléments du langage

- Attributs de tableaux :
  - ✓ **type vecteur is array (1 to 37) of integer;**
  - ✓ **type matrice is array (6 downto 2, 1 to 7) of real;**
  - ✓ **variable A : vecteur; variable B : matrice;**
  - ✓ left, right, high, low, range, reverse_range, lenght, ascending

- **A'left ===> 1** **A'right ===> 37**
- **A'high ===> 37** **A'low ===> 1**
- **A'length ===> 37** **A'range ===> 1 to 37**
- **A'reverse_range ===> 37 downto 1** **A'ascending ===> true**

- **B'left(1) ===> 6** **B'left(2) ===> 1**
- **B'right(1) ===> 2** **B'right(2) ===> 7**
- **B'low(1) ===> 2** **B'high(1) ===> 6**
- **B'low(2) ===> 1** **B'high(2) ===> 7**
- **B'length(1) ===> 5** **B'length(2) ===> 7**
- **B'range(1) ===> 16 downto 2** **B'reverse_range(1) ==> 2 to 6**
- **B'range(2) ===> 1 to 7** **B'reverse_range(2) ==> 7 downto 1**
- **B'ascending(1) ===> false** **B'ascending(2) ===> true**

5) Les éléments du langage

- Utilisation des attributs de tableaux

  - ✓ **type vecteur is array (1 to 37) of integer;**
  - ✓ **variable A : vecteur;**

  - ✓ **for i in A'range loop**

  - ✓ **if (A'left = 1) then**

Instructions concurrentes

6) Instructions concurrentes

- Les processus :
  - ils sont caractérisés par :
    - ✓ les signaux auxquels ils sont sensibles (liste de sensibilité)
    - ✓ les opérations séquentielles contenues
  - leur durée de vie d'un processus est égale au temps de simulation :
    - ✓ il ne se termine jamais
    - ✓ il peut s'endormir (wait)
  - toute instruction concurrente peut être traduite en processus (et notamment les affectations de signaux)
  - 2 possibilités :
    - ✓ liste de sensibilité
    - ✓ utilisation de l'instruction wait

6) Instructions concurrentes

```
label : process (liste signaux)
        déclarations
        begin
              instructions séquentielles
        end process label ;
```

Forme peu employée,
il s'agit des vestiges des
premières spécifications
du langage

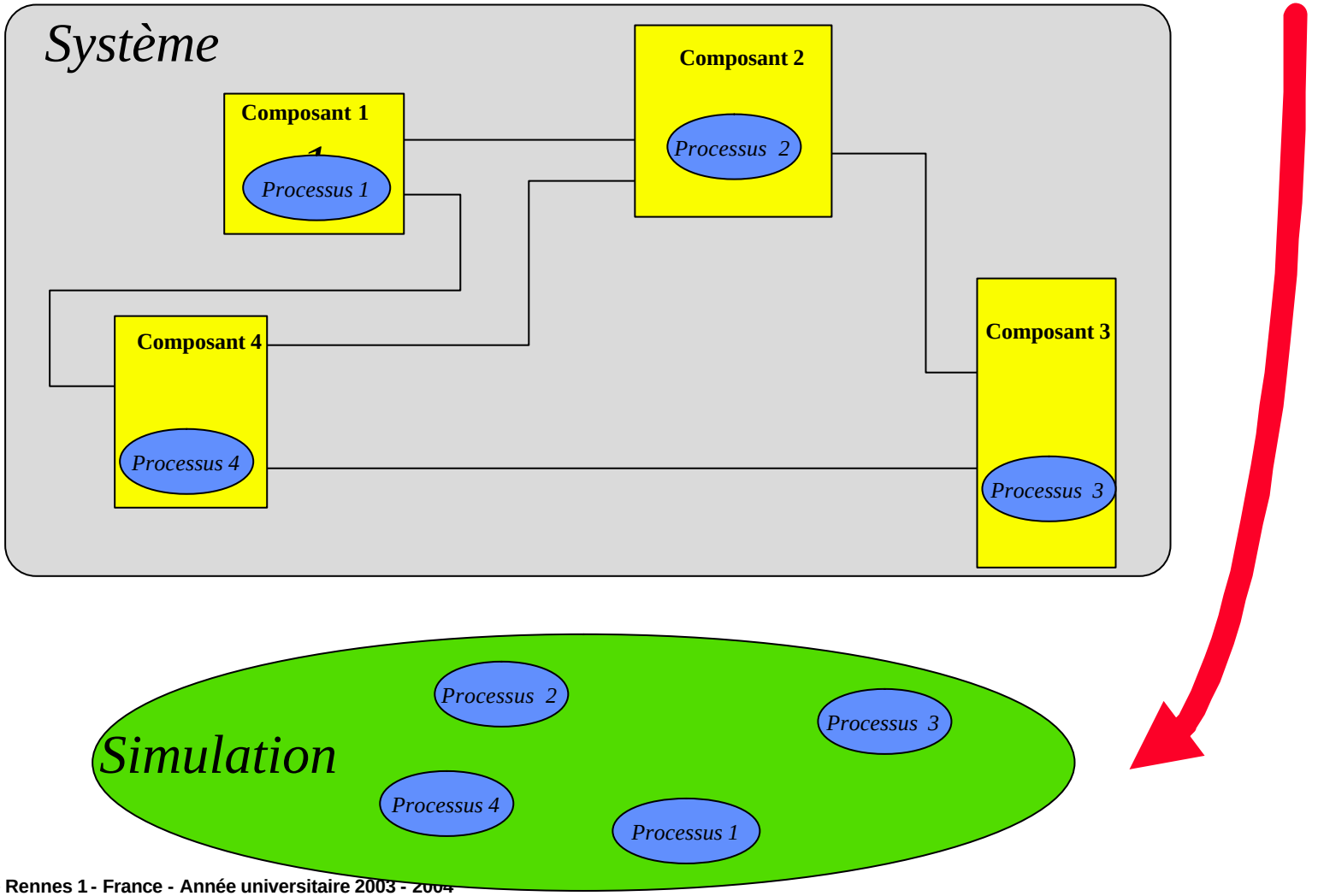
```
label : process
        déclarations
        begin
              instructions séquentielles
              ....
              wait on liste signaux ;
        end process label ;
```

Transformation de ce processus

```
label : process
        déclarations
        begin
              wait on liste signaux ;
              ....
              instructions séquentielles
        end process label ;
```

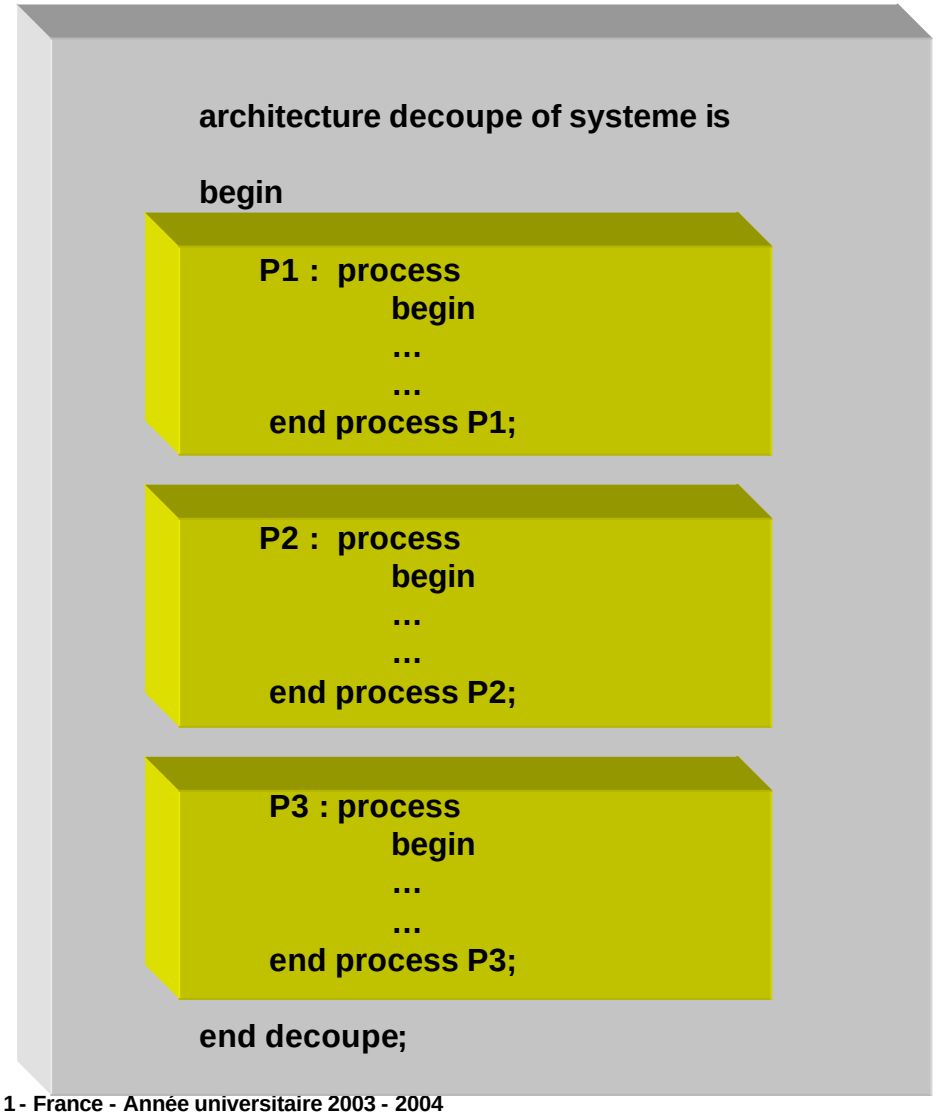
6) Instructions concurrentes

- dans un système assemblant des entités décrites de façon comportementale, les processus travaillent de façon concurrente



6) Instructions concurrentes

- ✓ plusieurs processus peuvent être présents dans une architecture

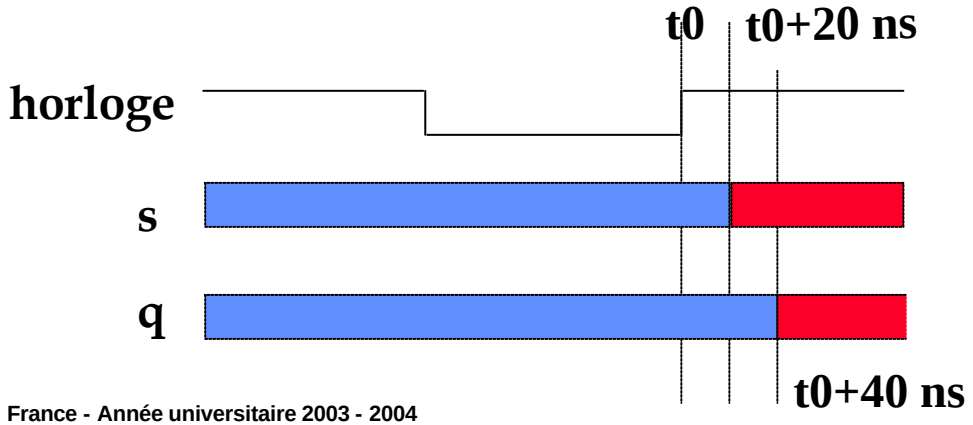


6) Instructions concurrentes

Les blocs :

- Pour réunir des instructions concurrentes :
  - ✓ partage de déclarations
  - ✓ garde d'affectation :
    - affectations soumises à une condition

```
bloc1 : bloc (horloge='1' and not horloge'stable)
begin
        s <= guarded valeur1 after 20 ns;
        q <= guarded valeur2 after 40 ns;
end bloc bloc1
```



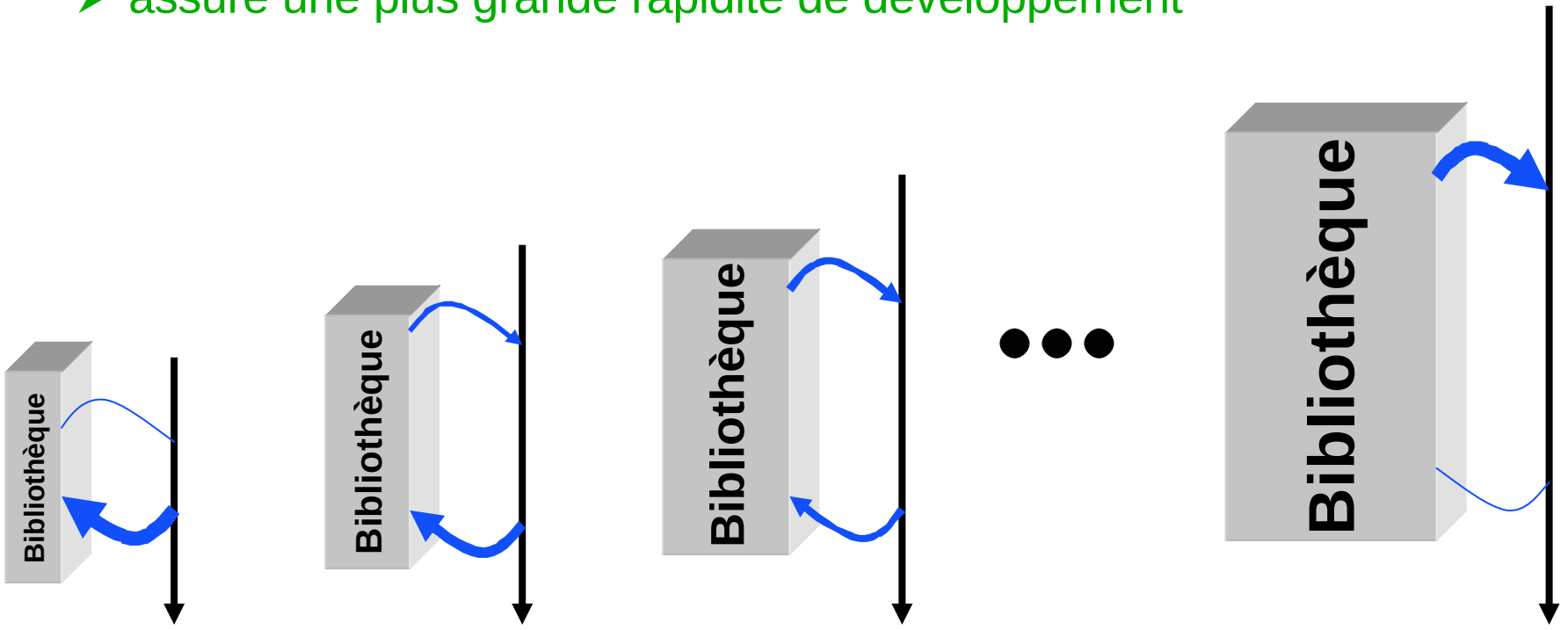
La généricité

7) La généricité

- C’est un moyen de transmettre une information à un bloc :
  - Vu de l’extérieur du bloc, la généricité == paramètre
  - Vu de l’intérieur du bloc, paramètres == constantes
- Intérêts :
  - description de composants généraux :
    - ✓ exemples :
      - registre N bits
      - additionneur à N entrées
  - permettre la réutilisation des composants :
    - ✓ description des bibliothèques par des modèles génériques

7) La généricité

- Intérêts de la généricité :
  - description de composants généraux
  - permettre la réutilisation des composants :
    - ✓ enrichissement progressif de la bibliothèque de travail
    - ✓ description de la bibliothèque par des modèles génériques
  - assure une plus grande rapidité de développement



7) La généricité

- Généralement, c'est l'entité qui est générique :
  - l'utilisation de générique dans la spécification d'entité

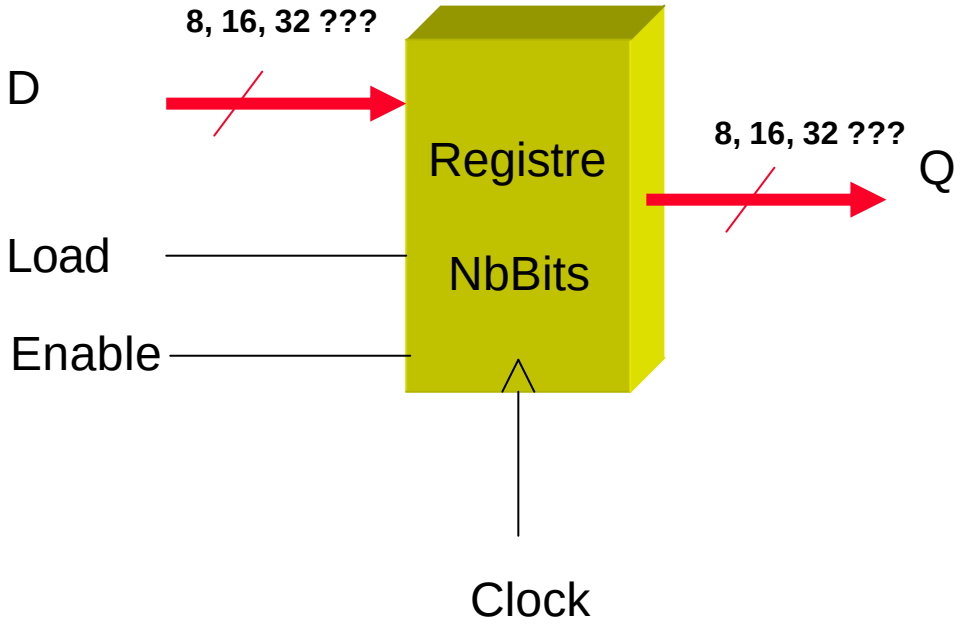
```
entity ……. is
        generic (…… ;
                  …….. ;
                  …….. );
        port (……;
                  ……;
                  ……..);
end ……... ;
```

- Mais l'architecture doit aussi être générique :
  - l'utilisation des attributs :
    - ✓ de tableaux
    - ✓ de types
    - ✓ de signaux

others, range, left, event, etc, etc

7) La généricité

- Généricité des entités :
  - un registre dont le nombre de bascules est générique



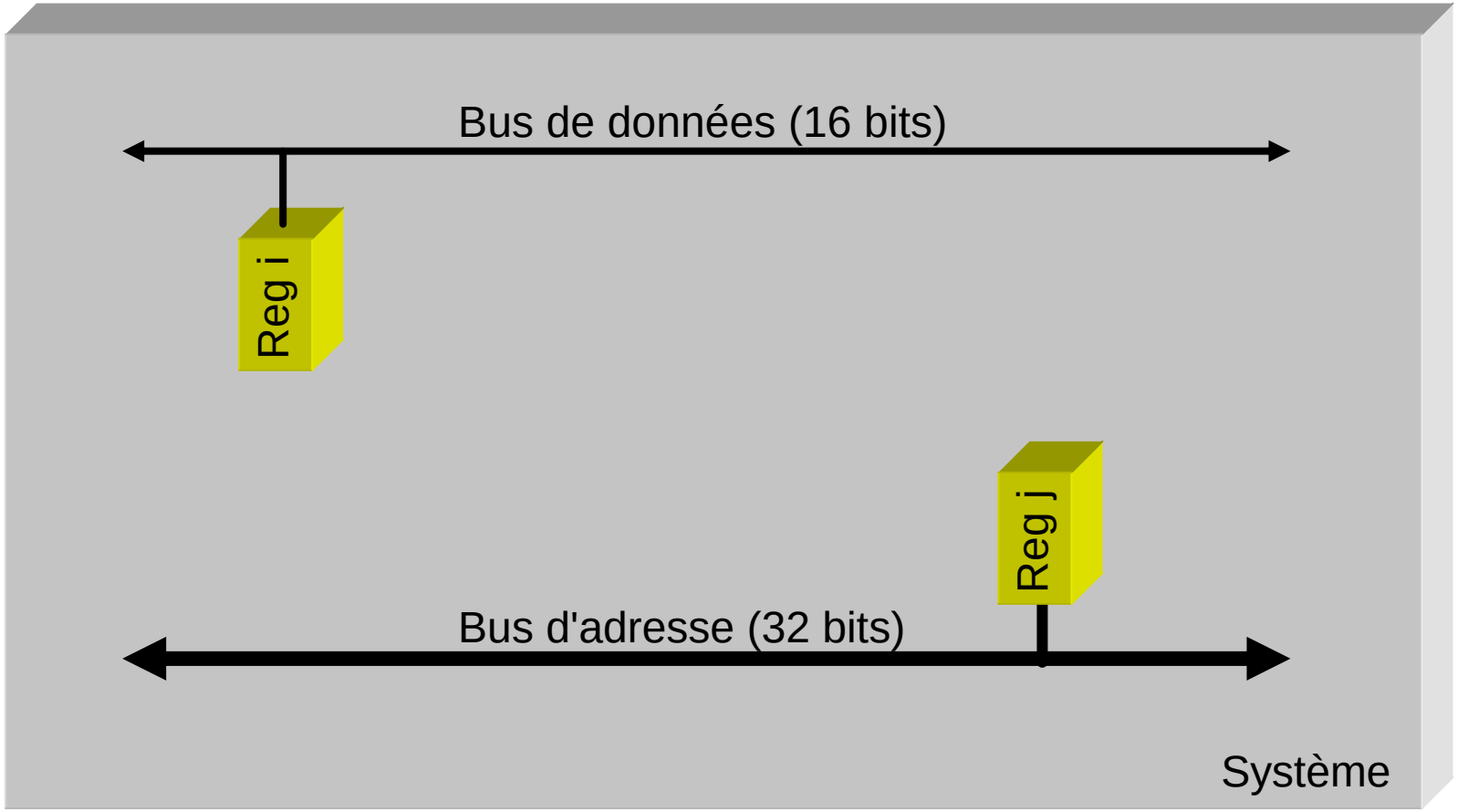
7) La généricité

- Description de l'entité

```
entity registre is
    generic (NbBits : INTEGER := 8);
    port (D : in std_logic_vector (NbBits-1 downto 0);
          Load : in std_logic ;
          Enable : in std_logic ;
          Clock : in std_logic ;
          Q : out std_logic_vector (NbBits-1 downto 0)
         );
end registre ;
```

7) La généricité

- Utilisation d'une entité générique :
  - ✓ soit le système suivant :
    - des registres sont connectés à des bus dont les tailles sont différentes



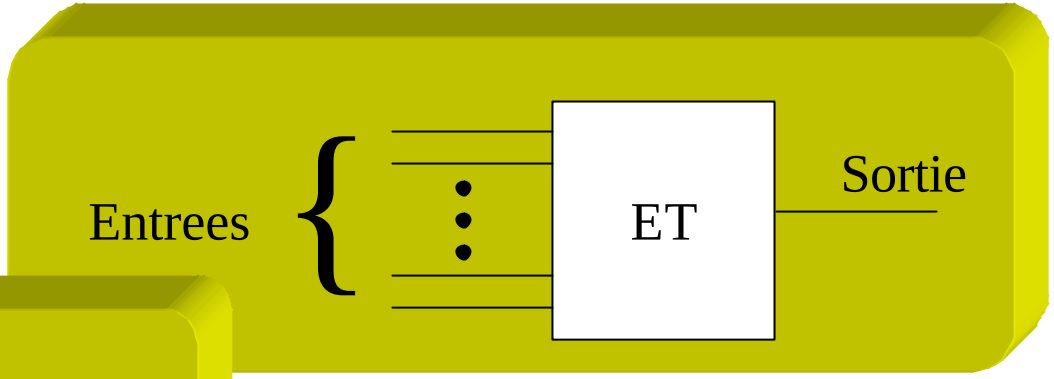
7) La généricité

- Description de l'architecture du système

```
architecture structure of Systeme is
        -- declaration des composants

        -- declaration des signaux
        signal BusData : std_logic_vector(15 downto 0);
        signal BusAddress : std_logic_vector(31 downto 0);
        signal SLoadi, SLoadj, SClock, SEnablei, SEnablej : std_logic;

begin
        …

        Regi : Registre
                generic map (16)
                port map (BusData, SLoadi, SEnablei, SClock, BusData);

        Regj : Registre
                generic map (32)
                port map (BusAddress, SLoadj, SEnablej, SClock, BusAddress);

        …
        …
end structure;
```

7) La généricité

- Autre exemple : un ET à N entrées



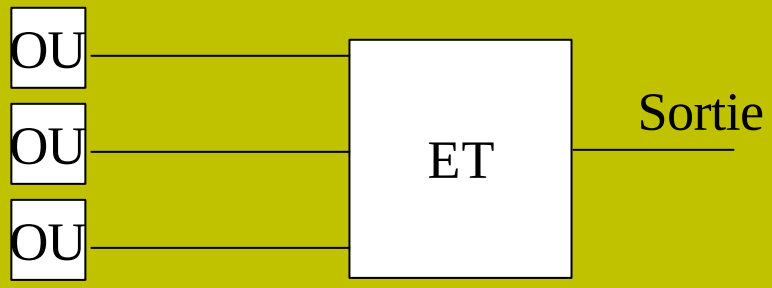

```
entity Et_N is
        generic ( N : Natural )
        port (      Entrees : in std_logic_vector ( 1 to N ) ;
                    sortie : out std_logic );
end Et_N ;
```

```
architecture comportement of Et_N is
begin
        process
                variable V : std_logic := '1' ;
        begin
                for i in 1 to N loop
                        V := V and Entrees (i) ;
                end loop ;
                Sortie <= V after 10 ns ;
                Wait on Entrees;
        end process ;
end comportement;
```

7) La généricité

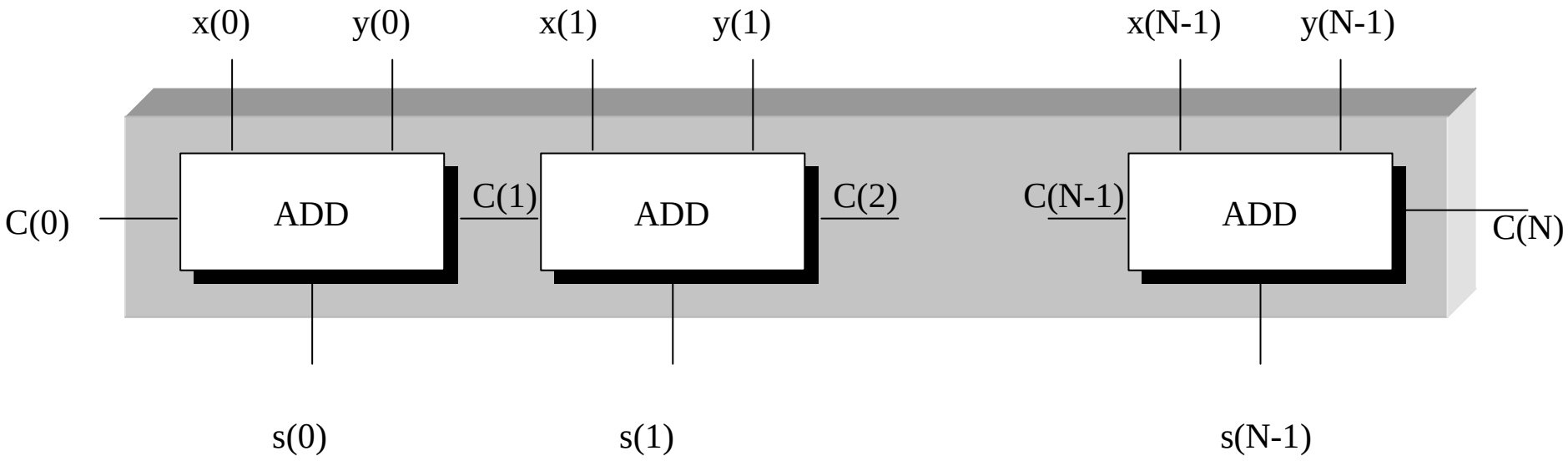
✓ Utilisation du ET générique :

```
architecture structure of systeme is
        component Et_N
                generic ( N : Natural )
                port (      Entrees : in std_logic_vector ( 1 to N ) ;
                            sortie : out std_logic );
        end component ;

        component OU
                port (      Entree : in std_logic ;
                            sortie : out std_logic );
        end component ;

        signal Entrees : std_logic_vector(2 downto 0);
        signal Sortie : std_logic;
        signal E0, E1, E2 : std_logic;
begin
        U0 : Et_N
                generic map(3)
                port map (Entrees, Sortie);
        U1 : OU port map( …, E1);
        U2 : OU port map( …, E2);
        U3 : OU port map( …, E3);

        Entrees(0) <= E1;
        Entrees(1) <= E2;
        Entrees(2) <= E3;
end comportement;
```



7) La généricité

- Autre exemple : additionneur structurelle générique :



- ✓ construit à partir d'un additionneur 1 bit
- ✓ assemblage des N additionneurs 1 bit afin de réaliser l'additionneur complet
- ✓ la valeur de N est inconnue avant l'instanciation du composant

7) La généricité

- On dispose de l'entité Add :

```
entity Add is
    port (
        A, B, Cin : in std_logic;
        S, Cout : out std_logic);
end Add ;
```

- L'entité Additionneur générique s'écrit :

```
entity AdditionneurN is
    generic (N : Natural := 8);
    port (
        X, Y : in std_logic_vector ( N-1 downto 0) ;
        Cin : in std_logic;
        S : out std_logic_vector (N-1 downto 0);
        Cout : out std_logic);
end AdditionneurN ;
```

```
architecture structurelle of  AdditionneurN is

    component Add
        port (
            A, B, Cin : in std_logic;
            S, Cout : out std_logic);
    end component;

    signal  C : std_logic_vector(0 to N);
begin
    for I in 0 to N-1 generate
        Instance : Add
            port map (X(I), Y(I), C(I), S(I), C(i+1));
    end generate;
    C(0) <= Cin;
    Cout <= C(N);
end structurelle ;
```

7) La généricité

Généricité par les attributs :

```
architecture comportement of registre is

begin
    process
        variable etat : std_logic_vector(NbBits -1 downto 0);
        begin
            if (Clock'event and Clock = '1') then
                if (Load = '1') then
                    etat := D;
                }
            }
            if (Enable = '1') then
                Q <= etat after 10 ns;
            else
                Q <= "ZZZZZZZZ" after 10 ns;
            end if;
            wait on Enable, Clock ;
        end process;

end comportement;
```

std_logic_vector(NbBits -1 downto 0);

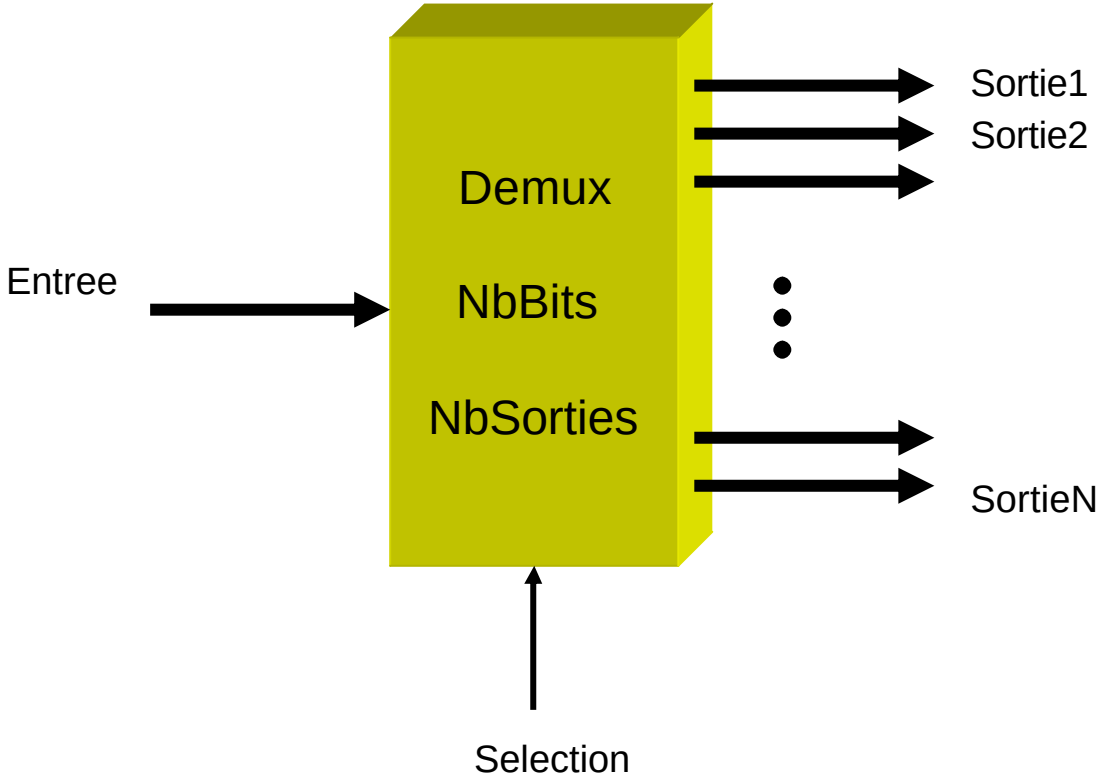
Pas très évolutif
(si la taille du registre est autre que 8 bits, la mise en haute impédance ne fonctionne plus)

Cette écriture assure un bon fonctionnement quelque soit la taille du registre

Q <= (others => 'Z') after 10 ns;

7) La généricité

- Autre exemple : le démultiplexeur



7) La généricité

- description de l'entité

```
entity Demux is
    generic (NbBits : Natural := 8;
          NbCmd : Natural := 4;
          NbSorties : Natural := 16);
    port (
          Entree : in std_logic_vector(NbBits -1 downto 0) ;
                   Selection : in std_logic_vector(NbCmd -1 downto 0);
                   Sorties : out std_logic_vector (NbSorties*NbBits -1 downto 0)
          );
end Demux ;

architecture comportement of Demux is

begin

          affectations : for i in      Selection'range      generate

                Sorties((i+1)*NbBits -1 downto i*NbBits) <=
                              Entree     when conv_positif(Selection) = i
                              else (others => '0');
          end generate affectations ;
end comportement;
```

Cette écriture assure une évolution de la taille du démultiplexeur

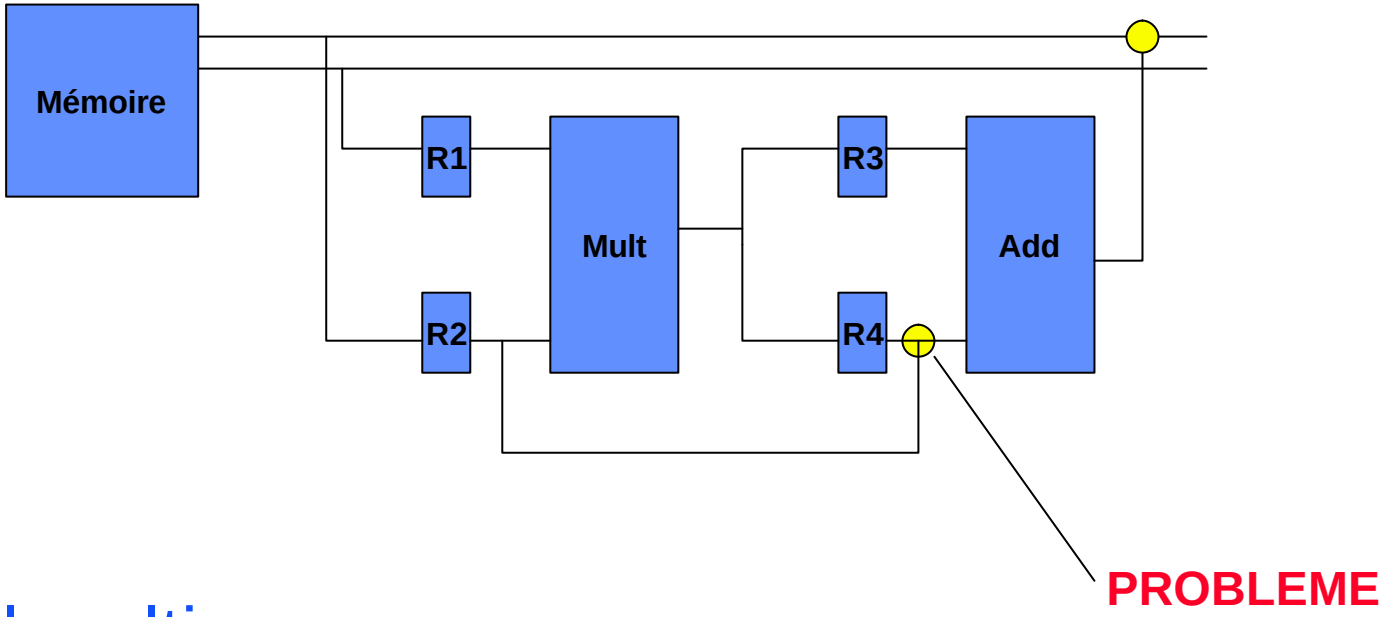
Ecriture non évolutive :

for i in 0 to 7 generate

La fonction de résolution

8) La fonction de résolution

- Quel est le problème ?
  - Soit le schéma suivant :



- Signal multi-sources

  Plusieurs *driver* pour le même signal

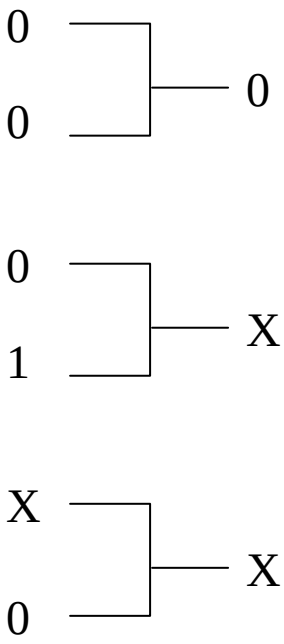
  (règle VHDl : un seul driver par signaux)

8) La fonction de résolution

- Définition d'une fonction :
  - prenant en compte les différentes sources du signal
  - ***calculant*** la ***valeur résolue*** du signal
- Nécessaire lorsque l'on veut simuler le système
- Caractéristiques de cette fonction :
  - appels réalisés (gérés) par le simulateur
    - ✓ pas d'appel explicite à la fonction
  - il n'y a qu'un seul paramètre d'entrée :
    - ✓ ce paramètre est toujours un tableau à 1 dimension non contraint

8) La fonction de résolution

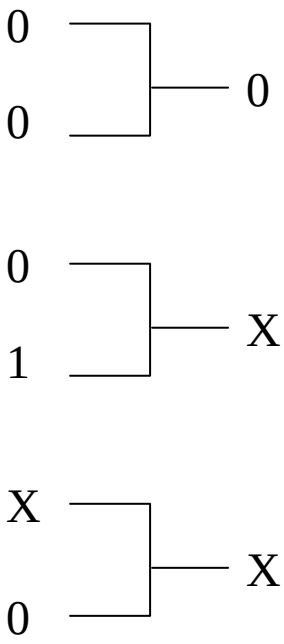
- Exemple :




```
TYPE  NewBit     IS ('Z','0','1','X') ;
TYPE TableauBits IS ARRAY (INTEGER RANGE <>) OF NewBit;
FUNCTION ResolutionBit4Etats (src : IN TableauBits) RETURN NewBit;
SUBTYPE Bit4Etats IS ResolutionBit4Etats NewBit;

FUNCTION ResolutionBit4Etats (src : IN TableauBits) RETURN NewBit IS
VARIABLE Result : NewBit := 'Z';
BEGIN
        FOR i IN src'RANGE LOOP
                CASE src(i) IS
                WHEN '0' =>
                        IF Result = '1' OR Result ='X' THEN
                            ASSERT false report "Conflit de signaux" SEVERITY ERROR;
                            RETURN 'X';
                        ELSE
                                Result := '0';
                        END IF;
                WHEN '1' =>
                        IF Result = '0' OR Result ='X' THEN
                            ASSERT false report "Conflit de signaux" SEVERITY ERROR;
                            RETURN 'X';
                        ELSE
                                Result := '1';
                        END IF;
                WHEN 'X' =>
                        Result := 'X';
                WHEN OTHERS =>
                END CASE ;
        END LOOP;
        RETURN Result;
END ResolutionBit4Etats ;
```

8) La fonction de résolution

- Exemple:



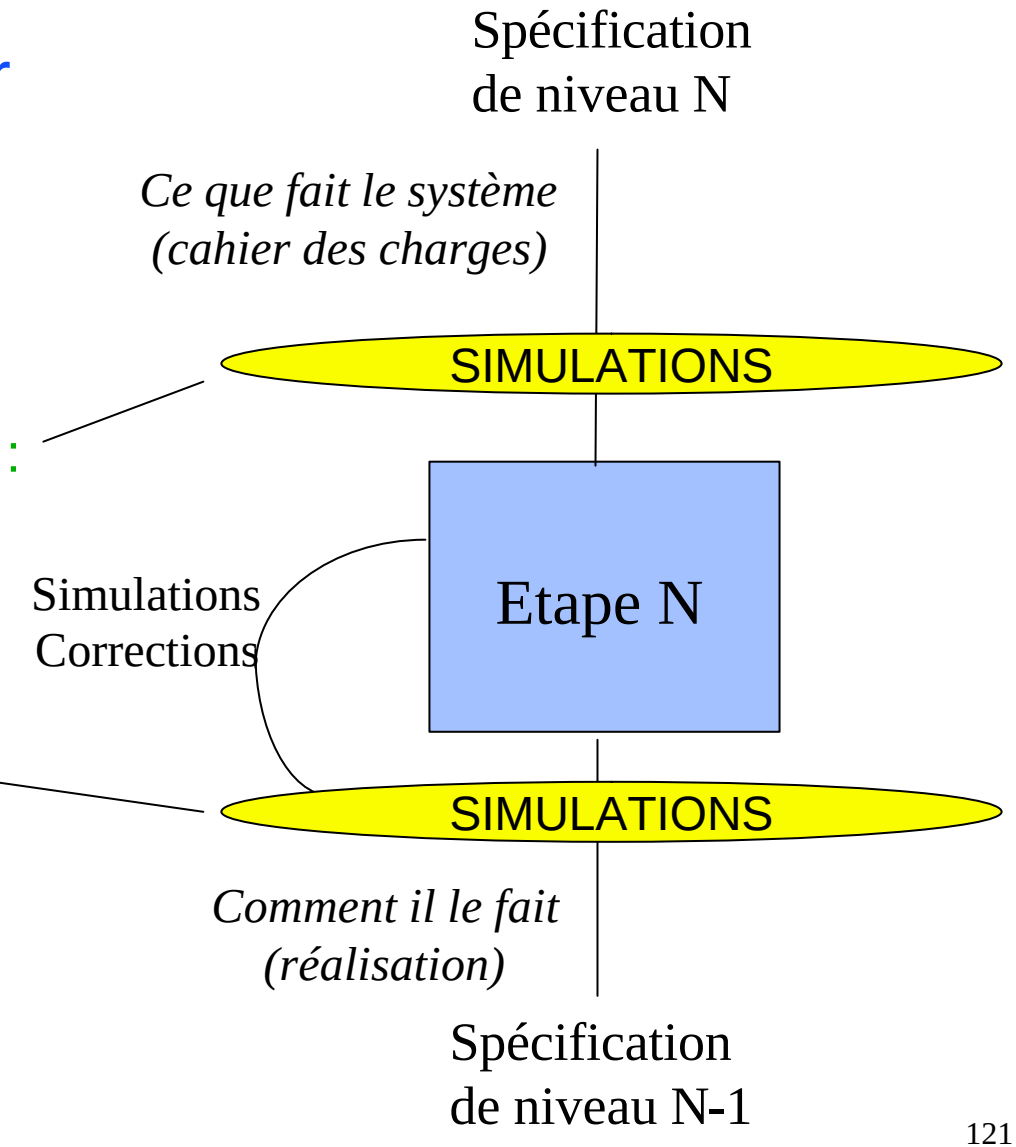

```
----Extrait du package std_logic_1164.vhd----------
    TYPE std_ulogic IS ( 'U','X','0', '1', 'Z', 'W', 'L', 'H', '-' );

    TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;

    CONSTANT resolution_table : stdlogic_table := (
--    ---------------------------------------------------
--    | U   X   0   1   Z   W   L   H   -     | |
--    ---------------------------------------------------
      ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ),  -- | U |
      ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ),  -- | X |
      ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ),      -- | 0 |
      ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ),      -- | 1 |
      ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ),    -- | Z |
      ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
      ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ),    -- | L |
      ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ),   -- | H |
      ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )     -- | - |
    );
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
   VARIABLE result : std_ulogic := 'Z';  -- weakest state default
   BEGIN
       IF    (s'LENGTH = 1) THEN    RETURN s(s'LOW);
         ELSE
           FOR i IN s'RANGE LOOP
              result := resolution_table(result, s(i));
           END LOOP;
        END IF;
     RETURN result;
 END resolved;
SUBTYPE std_logic IS resolved std_ulogic
```

Simulations et validations

9) Simulation et validation

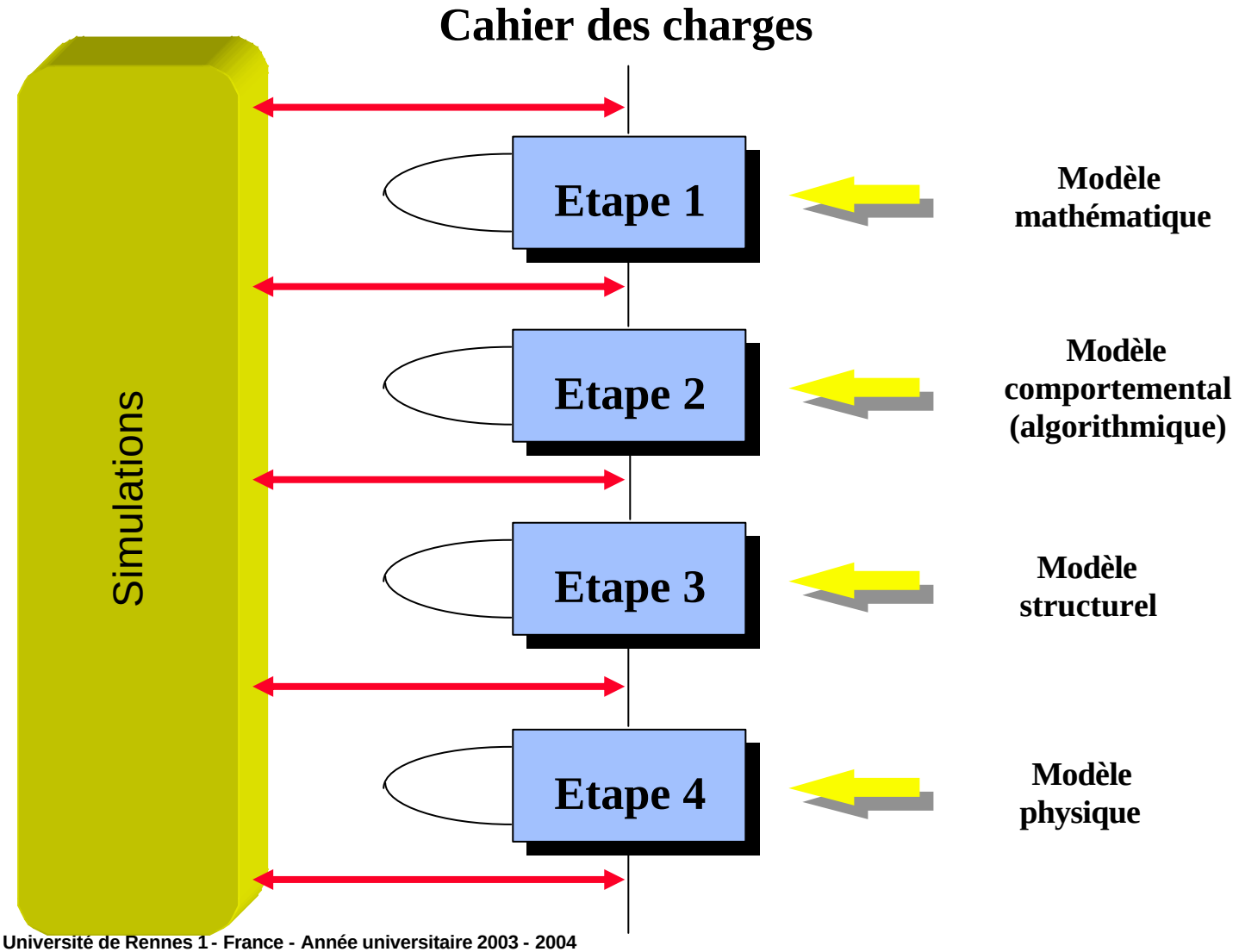
- Une bonne maîtrise de la conception passe par la **validation de chaque étape**

  - simulation comportementale : **REFERENCE**
  - simulation structurelle : **COMPARAISON AVEC LA REFERENCE**



9) Simulation et validation

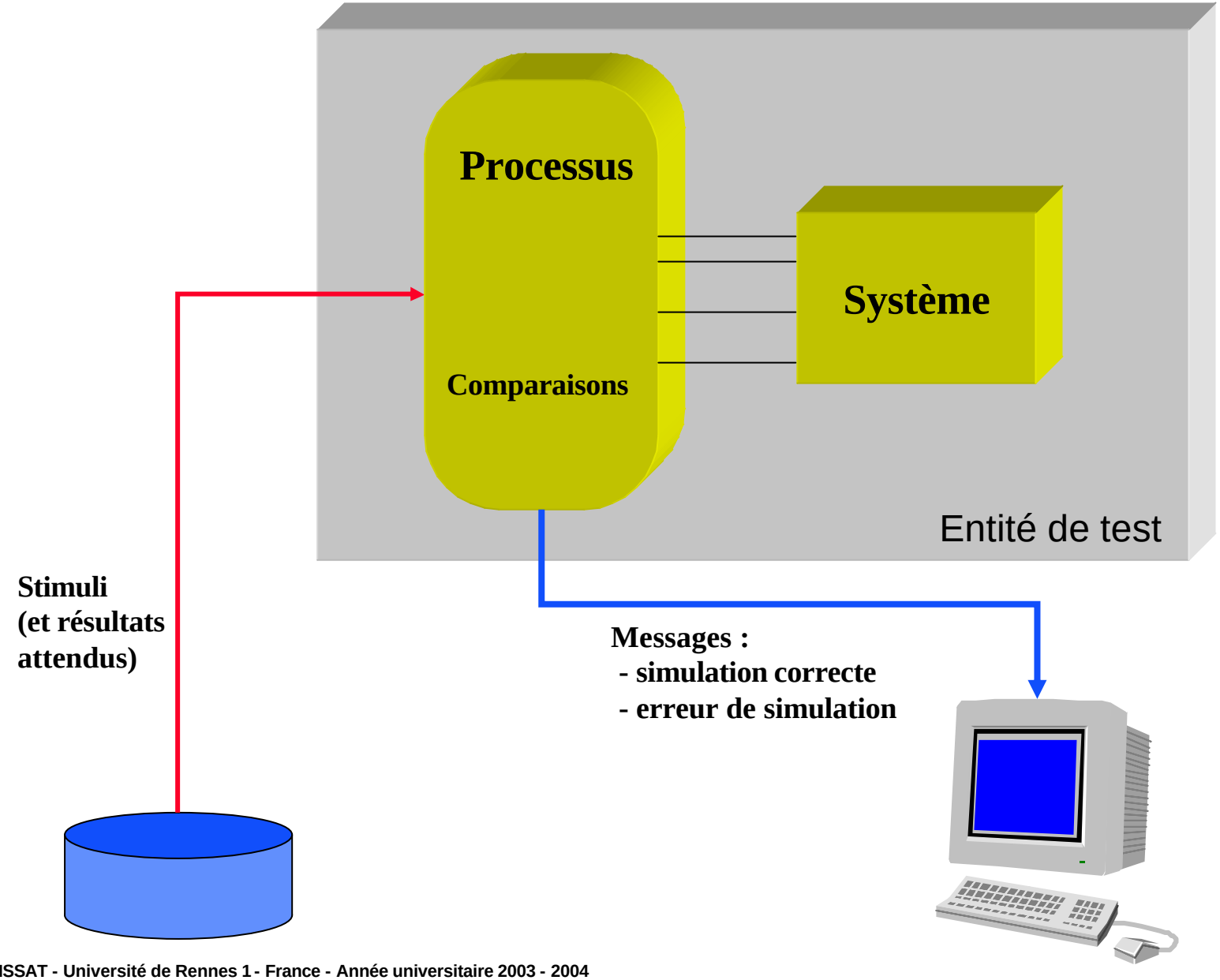
- **Simulation à chaque étape de la conception !!!!!!**



9) Simulation et validation

- Comment faire une simulation ?
  - instanciation du composant à tester
  - initialisation des signaux d'entrées
  - application d'une séquence de stimuli :
    - ✓ à partir d'un process et d'affectations des signaux d 'entrées
    - ✓ à partir d'un fichier contenant des vecteurs de test
  - analyse des résultats, analyse des transitions des sorties :
    - ✓ affichage des erreurs éventuelles

9) Simulation et validation

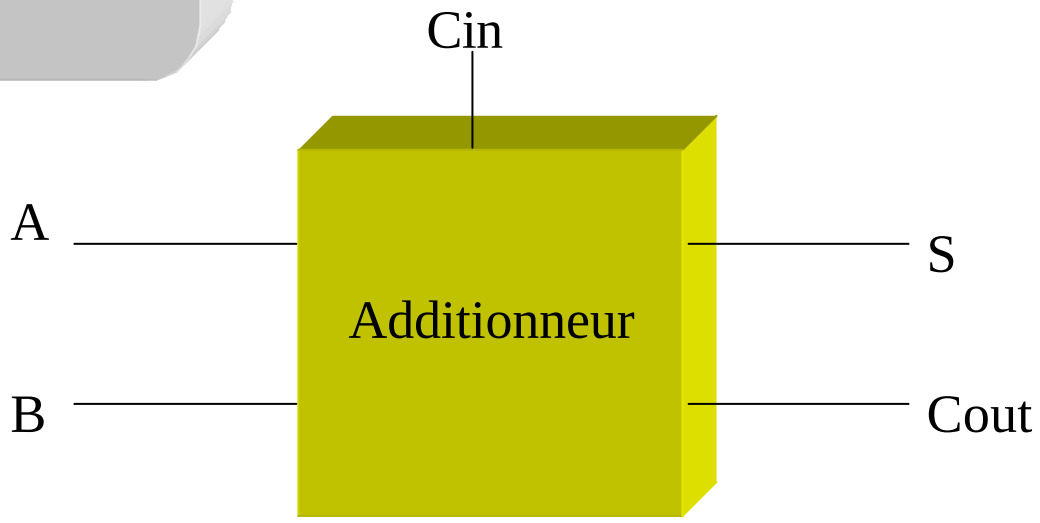


9) Simulation et validation

Description d'un composant de test : exemple

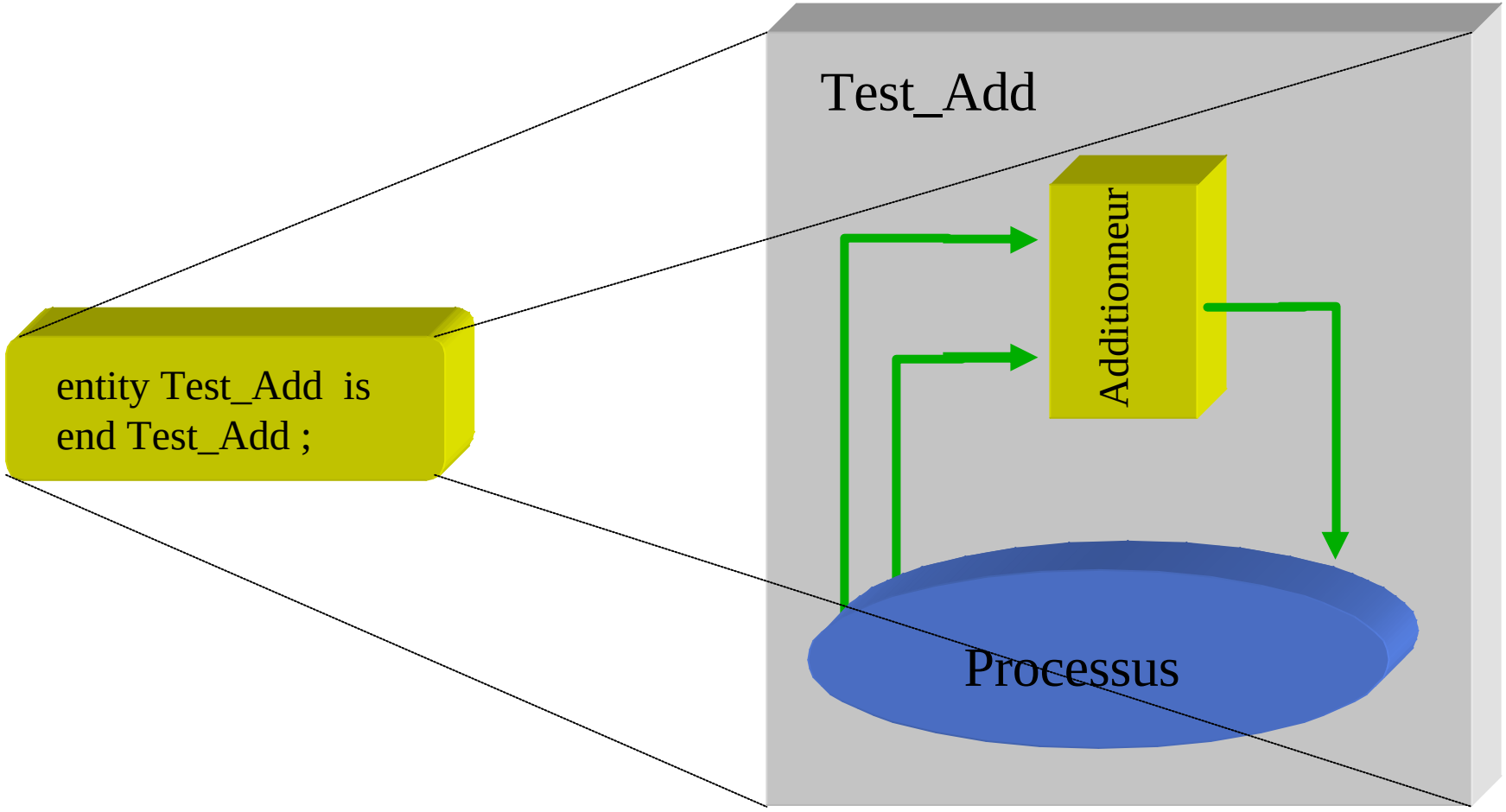
- Soit une entité additionneur à tester :

```
entity Additionneur is
        port (      A, B, Cin : in std_logic ;
                    S, Cout : out std_logic );
end Additionneur ;
architecture comportement of Additionneur is
...
end comportement;;
```



9) Simulation et validation

- On déclare une entité test englobant le composant de type Additionneur :
  - ✓ Vue externe du composant Test_Add
    - entité sans port d'entrées sorties, et sans générique



9) Simulation et validation

- Description de l'architecture de l'entité de test :
  - ✓ déclaration du composant à tester
  - ✓ déclaration des signaux d'entrées sorties
  - ✓ instanciation du composant
  - ✓ description d'un process générant les stimuli

```
architecture comportement of Test_Add  is

    -- Declaration du composant qui va etre utilise

    component Add
        port (
            A, B, CIn : in std_logic ;  S, Cout : out std_logic );
    end component ;


    -- Declaration des signaux d'entrée de  l'instance

    signal SA, SB, SOut, SCin , SCout : std_logic ;


    -- configuration

    for U1 : Add use entity Additionneur(comportement);

    constant TempsCycle : time := 10 ns ;
```

```
begin
    -- Instanciation du composant
    U1 : Additionneur port map (
        A => SA ,
        B => SB,
        S => SOut,
        Cin => SCin ,
        Cout => SCout
    );

    Simulation : process
        begin
            …
            …
            …
        end process;
end comportement;
```

9) Simulation et validation

- Description de l'architecture de l'entité de test : process de simulation
  - ✓ affectation des signaux d'entrées : règles :
    - toujours commencer par un wait for xx ns :
      - votre simulation commence réellement au temps xx ns;
      - évite les problèmes liés aux initialisations effectuées par le simulateur
    - toujours terminer la simulation un par wait :
      - évite les rebouclages du process sur lui même

```
Simulation : process
     begin
                    wait for TempsCycle ns;

                    SA <= '0';
                    SB <= '0';
                    SCin <= '0';
                    wait for TempsCycle ns;

                    SA <= '0';
                    SB <= '0';
                    SCin <= '1';
                    wait for TempsCycle ns;

                    ...
                    ...
                    SA <= '1';
                    SB <= '1';
                     SCin <= '1';

                    wait ;
     end process;
```

9) Simulation et validation

- Description de l'architecture de l'entité de test : process de simulation
  - ✓ affectation des signaux d'entrées par une boucle :

```
architecture comportement of Test_Add  is

      …
      …

     constant TempsCycle : time := 10 ns ;
     type TableauEntrees is array (0 to 2) of std_logic;
     type TableauVecteur is array (0 to 7) of TableauEntrees;

     constant Vecteur : TableauVecteur :=
          (
                (0, 0, 0),
                (0, 0, 1),
                (0, 1, 0),
                (0, 1, 1),
                (1, 0, 0),
                (1, 0, 1),
                (1, 1, 0),
                (1, 1, 1)
          );

    signal SA, SB, SOut, SCin, SCout : std_logic ;


begin
```

```
-- Instanciation du composant
      U1 : Additionneur port map (…);


Simulation : process
      begin
            wait for TempsCycle ;

            for i in Vecteur'range(1) loop
                  SA <= Vecteur(i)(0);
                  SB <= Vecteur(i)(1);
                  SCin <= Vecteur(i)(2);
                  wait for TempsCycle;
            end loop;

            wait ;
end process;
```

9) Simulation et validation

- Description de l'architecture de l'entité de test : process de simulation
  - ✓ affectation des signaux d'entrées par une boucle
  - ✓ analyse des résultats

```
architecture comportement of Test_Add  is

      …
      …

    constant TempsCycle : time := 10 ns ;
    type TableauEntrees is array (0 to 4) of std_logic;
    type TableauVecteur is array (0 to 7) of TableauEntrees;

    constant Vecteur : TableauVecteur :=
        (
            (0, 0, 0, 0, 0),
            (0, 0, 1, 0, 1),
            (0, 1, 0, 0, 1),
            (0, 1, 1, 1, 0),
            (1, 0, 0, 0, 1),
            (1, 0, 1, 1, 0),
            (1, 1, 0, 1, 0),
            (1, 1, 1, 1, 1)
        );

    signal SA, SB, SOut, SCin, SCout : std_logic ;


begin
```

```
-- Instanciation du composant
      U1 : Additionneur port map (…);

Simulation : process
      begin
            wait for TempsCycle ;

            for i in Vecteur'range(1) loop
                  SA <= Vecteur(i)(0);
                  SB <= Vecteur(i)(1);
                  SCIn <= Vecteur(i)(2);
                  wait for TempsCycle;
                  assert (SCout = Vecteur(i)(3))
                              report "Probleme sur la sortie Cout"
                              severity warning;
                  assert (SOut = Vecteur(i)(4))
                              report "Probleme sur la sortie S"
                              severity warning;
            end loop;

            wait ;
end process;
```

9) Simulation et validation

- Description de l'architecture de l'entité de test : process de simulation
  - ✓ lecture des stimuli dans un fichier
  - ✓ affectation des signaux d'entrées par une boucle analyse des résultats

```
architecture comportement of Test_Add  is

      …
      …

     signal SA, SB, SOut, SCin, SCout : std_logic ;


begin

      -- Instanciation du composant
      U1 : Additionneur port map (…);


Simulation : process
      file VecteursIN  : integer is in "VecteursIN ";
      file VecteursOut : integer is in "VecteursOUT";
      variable ligne : line;
      variable VA, VB, VCin, VCout, VOut : integer;
      variable Tps : integer;
      variable TempsCycle : time := 10 ns;

begin
            wait for TempsCycle ;

            readline(VecteursIN , ligne);
            read(ligne, Tps);
            TempsCycle = Tps ns;

            while not endfile(VecteursIn ) loop
                  read(ligne, VA);
                  read(ligne, VB);
                  read(ligne, VCin);
                  read(ligne, VCout);
                  read(ligne, VOut);

                  SA <= Integer2Bit(VA);
                  SB <= Integer2Bit((VB);
                  SCIn <= Integer2Bit(VCin);
                  wait for TempsCycle;
                  assert (SCout = Integer2Bit(VCout))
                              report "Probleme sur la sortie Cout"
                              severity warning;
                  assert (SOut = Integer2Bit(VOut)
                              report "Probleme sur la sortie S"
                              severity warning;
            end loop;

            wait ;
end process;
```

9) Simulation et validation

- Test des composants synchrone à une horloge :
  - placer un process générant l'horloge à côté du process de simulation

```
entity Test_Registre
end Test_Registre;


architecture comportement of Test_Registre  is

      -- Declaration du composant qui va etre utilise

      component Registre
            port (
                  D, clock,  : in std_logic ;  Q : out std_logic );
           end component ;


      -- Declaration des signaux d'entrée de  l'instance

      signal SD, SClock, SQ : std_logic;


      -- configuration

      for U1 : Registre use entity Registre(comportement );
```

```
begin
      -- Instanciation du composant
      U1 : Registre port map (
                  SD, SClock, SQ
          );

      Simulation : process
            begin
                  …
                  …
                  ...
            end process Simulation;

      Horloge : process
            begin
                  SClock <= '0' ;
                  wait for 50 ns;
                  SClock <= '1' ;
                  wait for  50 ns;
          end process Horloge ;

end comportement;
```

Exemples :

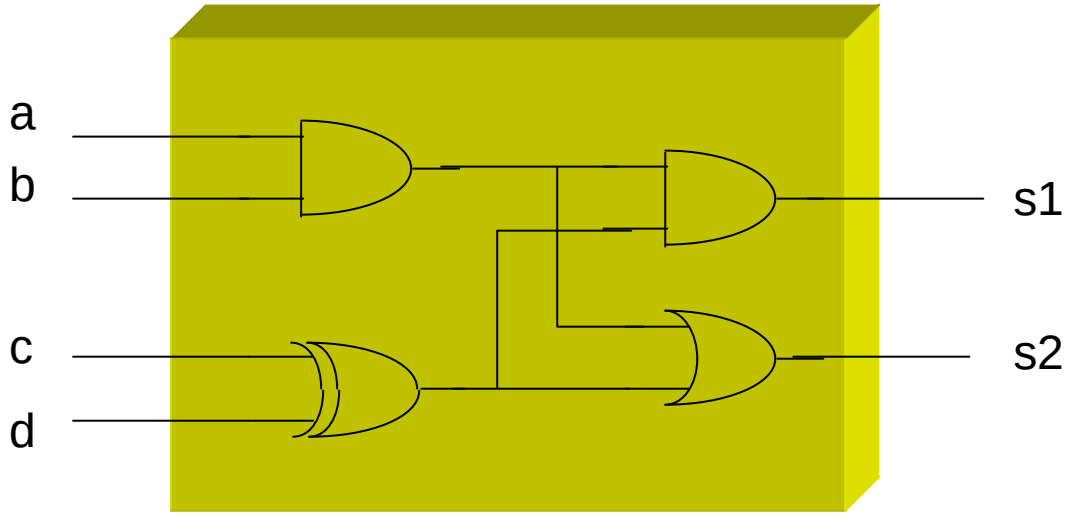
- logique combinatoire

- logique séquentielle

10) Logique combinatoire

Exemple simple :

- soit la fonction :



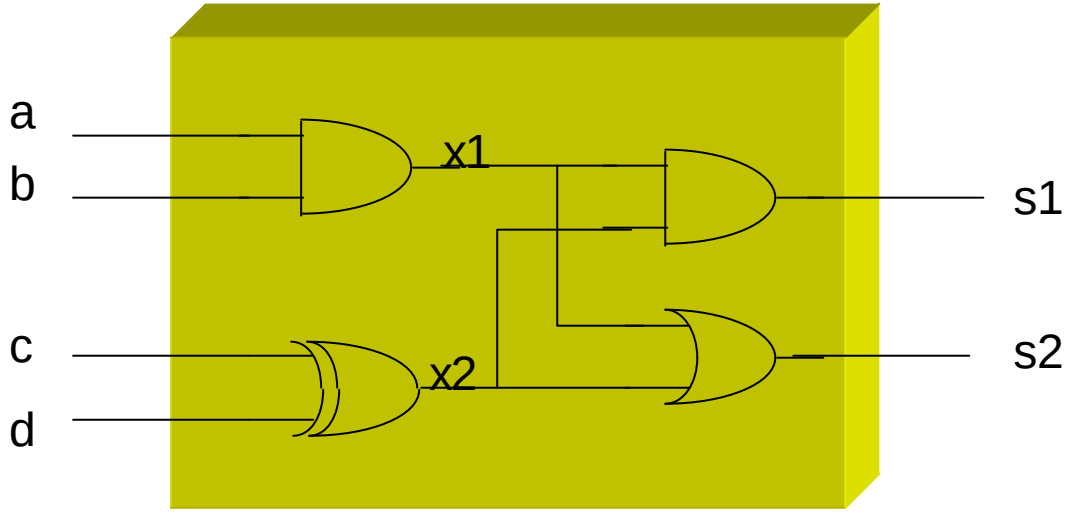

```
ENTITY Fonc IS
    PORT   ( a  : IN  std_logic;
             b  : IN std_logic;
             c  : IN std_logic;
             d  : IN std_logic;
             s1  : OUT std_logic;
             s2  : OUT std_logic)  ) ;
END Fonc ;
```

```
ARCHITECTURE flot1 OF Fonc IS

BEGIN
   s2 <= (a and b) or (c xor d) after
                max(Tand + Tor, Txor + Tor);
   s1 <= (a and b) and (c xor d) after
                max(Tand + Tand, Tand + Txor);
END flot1 ;
```

```
ARCHITECTURE flot2 OF Fonc IS

BEGIN
   s1 <= (a and b) and (c xor d) after
                max(Tand + Tand, Tand + Txor);
   s2 <= (a and b) or (c xor d) after
                max(Tand + Tor, Txor + Tor);
END flot2 ;
```

10) Logique combinatoire

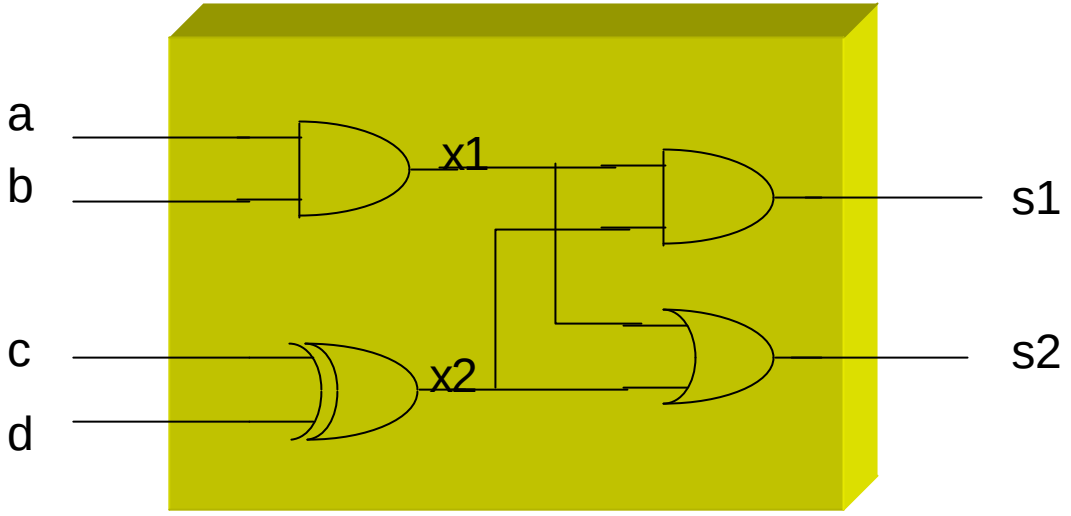
- Autre description :
  - ✓ introduction des signaux intermédiaires




```
ARCHITECTURE flot3 OF Fonc IS

BEGIN
   s1 <= (x1 and x2) after Tand;
   s2 <= (x1 or x2) after Tor;
   x1 <= (a and b) after tand;
   x2 <= (c xor d) after Txor;
END flot3 ;
```

10) Logique combinatoire

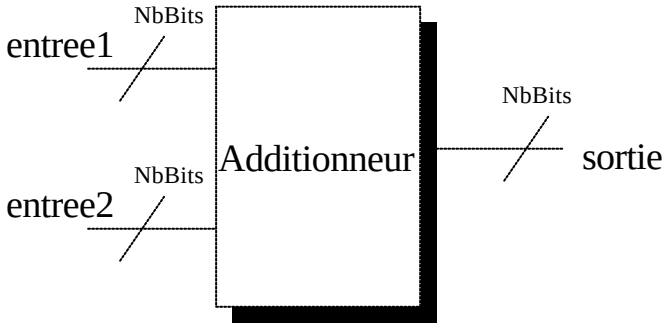
- Et encore une autre ...




```
ARCHITECTURE flot4 OF Fonc IS

BEGIN
   s1 <= '1' when ((a and b) and (c xor d))  else '0';
   s2 <= '1' when ((a and b) or (c xor d)) else '0';
END flot4 ;
```

10) Logique combinatoire

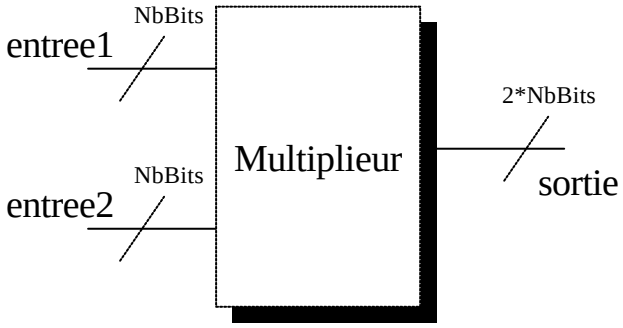
- Additionneur :
  - généricité sur :
    - ✓ le nombre de bits
    - ✓ le temps de calcul




```
ENTITY Additionneur IS
        GENERIC (Tadd : TIME ;
                 NbBits : INTEGER   );
        PORT    (       entree1  : IN  std_logic_vector(NbBits-1 DOWNTO 0) ;
                        entree2  : IN std_logic_vector(NbBits-1 DOWNTO 0)  ;
                        sortie   : OUT std_logic_vector(NbBits-1 DOWNTO 0)  ) ;
END Additionneur ;

ARCHITECTURE comportementale OF Additionneur IS

BEGIN
        ProcessAdditionneur : PROCESS
        VARIABLE e1, e2, s : INTEGER;

        BEGIN
                e1 := Conv_Integer (entree1) ;
                e2 := Conv_Integer (entree2) ;
                s := e1 + e2 ;
                sortie <= Conv_Std_Logic_Vector (s, NbBits) After Tadd ;
                wait on entree1, entree2;
        END PROCESS ProcessAdditionneur ;
END comportementale ;
```

10) Logique combinatoire

- Multiplieur :
  - généricité sur :
    - ✓ le nombre de bits
    - ✓ le temps de calcul

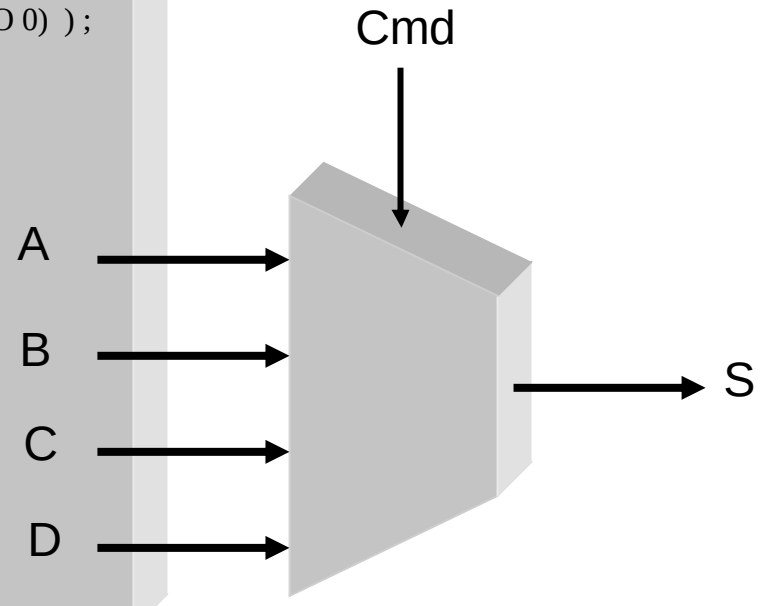



```
ENTITY Multiplieur IS
        GENERIC (Tmult : TIME ;
                 NbBits : INTEGER   );
        PORT    (      entree1  : IN  std_logic_vector(NbBits-1 DOWNTO 0) ;
                       entree2  : IN std_logic_vector(NbBits-1 DOWNTO 0)      ;
                       sortie   : OUT std_logic_vector(2*NbBits-1 DOWNTO 0)  ) ;
END Multiplieur ;

ARCHITECTURE comportementale OF Multiplieur IS

BEGIN
        ProcessMultiplieur : PROCESS
        VARIABLE e1, e2, s : INTEGER;

        BEGIN
                e1 := Conv_Integer (entree1) ;
                e2 := Conv_Integer (entree2) ;
                s := e1 * e2 ;
                sortie <= Conv_Std_Logic_Vector (s, 2*NbBits) After Tmult ;
                wait on entree1, entree2;
        END PROCESS ProcessMultiplieur ;
END comportementale ;
```

10) Logique combinatoire

- Exemple du multiplexeur :

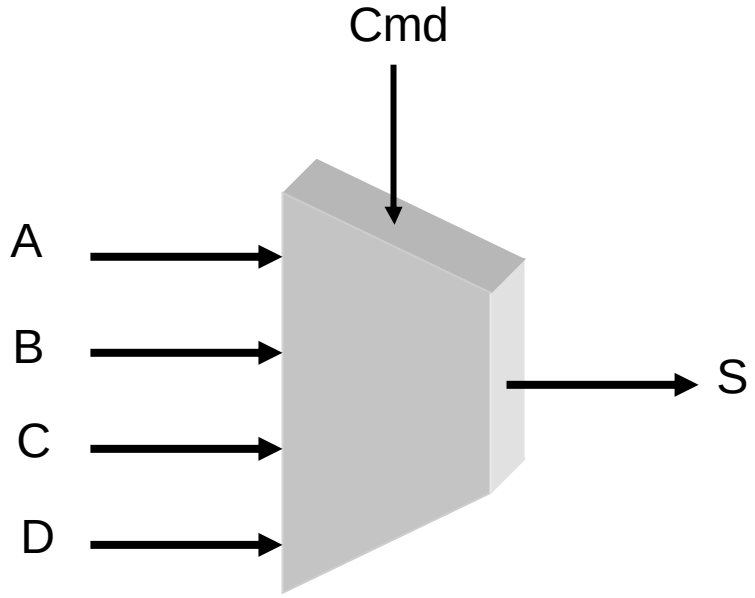
```
ENTITY Multiplexeur IS
        GENERIC (Tmux : TIME ;
                 NbBits : INTEGER   );
        PORT    (       A  : IN  std_logic_vector(NbBits-1 DOWNTO 0) ;
                        B  : IN std_logic_vector(NbBits-1 DOWNTO 0)      ;
                        C  : IN std_logic_vector(NbBits-1 DOWNTO 0)      ;
                        D  : IN std_logic_vector(NbBits-1 DOWNTO 0)      ;
                        cmd  : IN std_logic_vector(1 DOWNTO 0)      ;
                        S  : OUT std_logic_vector(NbBits-1 DOWNTO 0)  ) ;
END Multiplexeur ;


ARCHITECTURE flotdedonnees1 OF Multiplexeur IS

BEGIN

        S <=            A  WHEN (cmd = "00")
                ELSE    B  WHEN (cmd = "01")
                ELSE    C  WHEN (cmd = "10")
                ELSE    D;

END flotdedonnees ;
```



10) Logique combinatoire

Exemple du multiplexeur : autre description

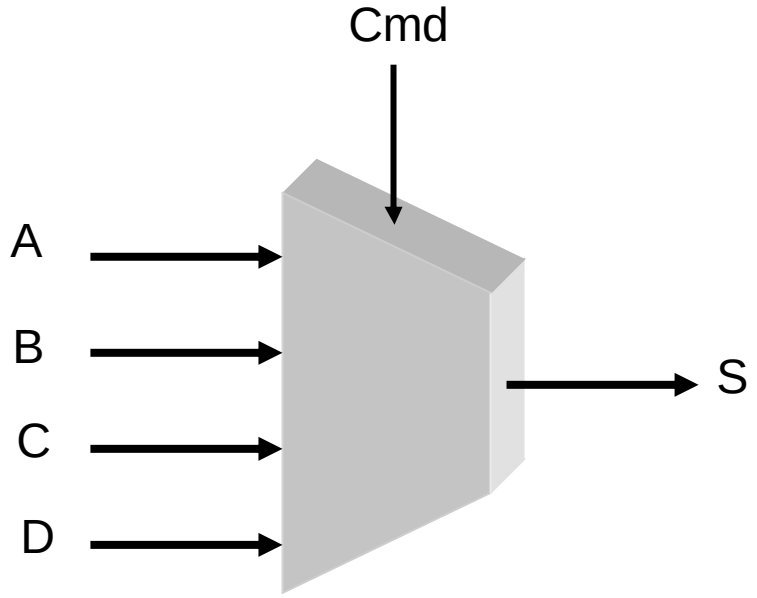
```
ARCHITECTURE flotdedonnees2 OF Multiplexeur IS

BEGIN
        WITH cmd SELECT
        S <=        A  WHEN (cmd = "00")
                    B  WHEN (cmd = "01")
                    C  WHEN (cmd = "10")
                    D  WHEN OTHERS;

END flotdedonnees ;
```



10) Logique combinatoire

- Exemple du multiplexeur : autre description

```
ARCHITECTURE comportementale OF Multiplexeur IS

BEGIN
        PROCESS
        BEGIN
                if cmd = "00" THEN
                        S <= A ;
                ELSIF cmd = "01" THEN
                        S <= B;
                ELSIF cmd = "10" THEN
                        S <= C;
                ELSIF cmd = "11" THEN
                        S <= D;
                END IF;
                WAIT ON cmd, A, B, C, D;
        END PROCESS;
END flotdedonnees ;
```



10) Logique combinatoire

- Le décodeur :

```
entity decodeur is
        generic (
                NbSorties : integer;
                Log2NbSorties : integer
        ) ;
        port (
                sorties : out std_logic_vector (NbSorties -  1 downto 0) ;
                cmd : in std_logic_vector (log2NbSorties -1 downto 0)
        );
end decodeur;

architecture flot of decodeur is

begin
        affectations : for i in sorties'range generate
                sorties(i) <= '1' when conv_positif(cmd) = i
                                        else '0';
        end generate affectations;
end flot ;
```

10) Logique combinatoire

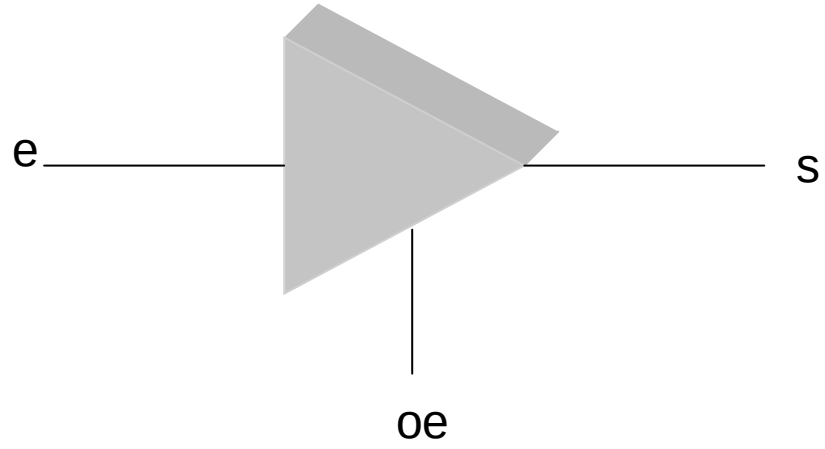
- Réalisation d'une fonction logiques complexes :
  - passage par la table de vérité :

A	B	C	D	S
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	..
..	..	..	..	
..	..	..	..	

```
type TABLE1 is array (0 to 15) of std_logic;

constant codage1 : TABLE1 := (
                '1',
                '1',
                '1',
                '0',
                '0',
                '0',
                …
                …
);

V:= 8 * conv_integer (A) + 4 * conv_integer (B)
    + 2 * conv_integer (C) + 1 * conv_integer (D) ;
S <= codage1 (V) ;
```

10) Logique combinatoire

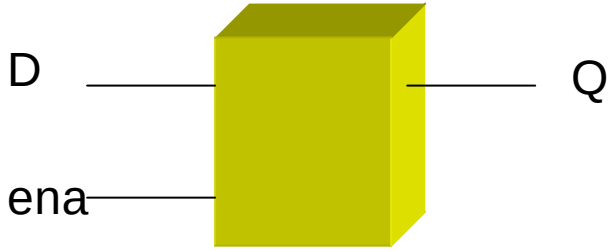
- Module de mise en haute impédance :




```
entity tristate is
        port (
                e : in std_logic;
                oe : in std_logic;
                s : out std_logic
        );
end tristate ;

architecture flot of tristate is

begin
        s <= e when oe = '1' else 'Z';
end flot ;
```

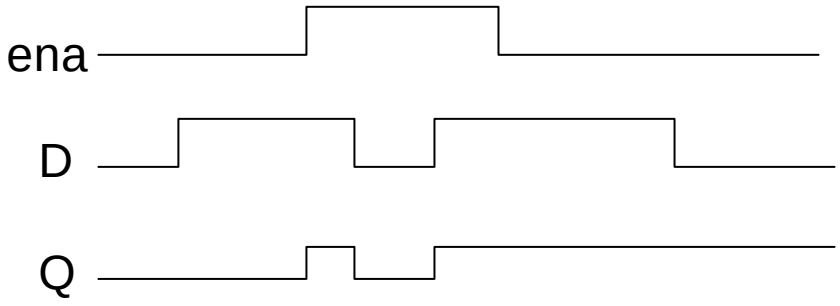
10) Logique séquentielle

Bascule D Latch



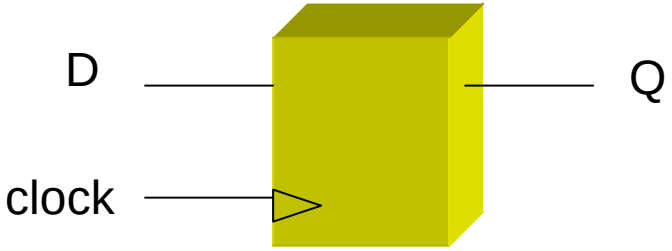

```
entity BasculeD is
        port (      d, ena : in std_logic;
                    Q : out std_logic);
end BasculeD ;
```

```
architecture comp of BasculeD is
begin
   process
   begin
      if ena = '1' then
            test <= D;
      else
            test <= test;
      end if;
      wait on ena, D;
   end process;

   Q <= test;
end comp;
```



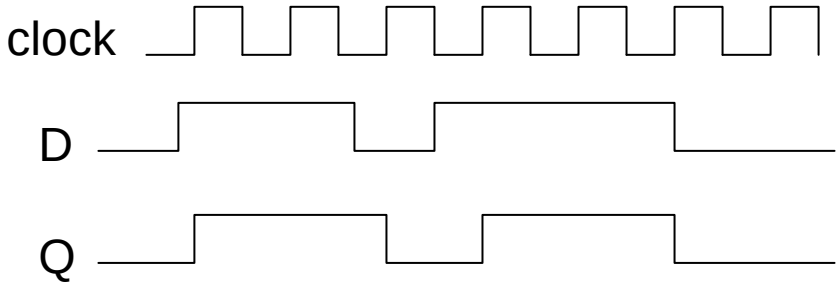
10) Logique séquentielle

Bascule D edge triggered (synchrone)




```
entity BasculeD is
        port (      d, clock : in std_logic;
                    Q : out std_logic);
end BasculeD ;
```

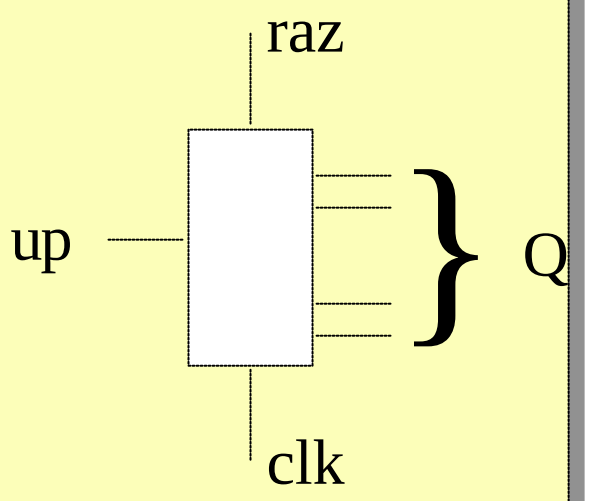
```
architecture comp of BasculeD is
begin
   process
   begin
      if cloc'event and clock = '1' then
            test <= D;
      else
            test <= test;
      end if;
      wait on clock;
   end process;

   Q <= test;
end comp;
```



10) Logique séquentielle

- Compteur de taille quelconque :

```
entity Compteur is
        generic (taille : integer ; Tps : Time);
        port (      clk, raz, up : in Std_logic;
                    Q : out Std_logic_vector(taille - 1  downto 0));
end Compteur;

architecture comportement of Compteur is
        signal etat : Std_logic_vector(taille-1  downto 0);
begin
        process
        begin
        if (raz ='0') then                  etat <= (others => '0') ;
        elsif (clk'event and clk = '1') then
                    if (up = '1') then      etat <= etat + 1;
                    else                    etat <= etat - 1;
                    end if;
        end if;
        wait on clk, raz, up;
        end process;
        Q <= etat  after Tps;
end comportement;
```



10) Logique séquentielle

Bascules D avec remise à zéro asynchrone ou synchrone

```
entity BDA is
        port (      d, raz, clk : in std_logic;
                    Q : out std_logic);
end BDA;

architecture comp of BDA is
begin
   process
   begin
      if (raz = '0')  then
         Q <= '0' ;
      elsif clk'event and clk = '1' then
         Q <= D ;
      end if;
      wait on clk, raz;
   end process;
end comp;
```

Asynchrone

```
entity BDS is
        port (      d, raz, clk : in std_logic;
                    Q : out std_logic);
end BDS;
architecture comp of BDS is
begin
   process
   begin
      if clk'event and clk = '1'  then
         if raz = '0' then
            Q <= '0' ;
         else
            Q <= D;
         end if;
      end if;
      wait on clk;
   end process;
end comp;
```

Synchrone

10) Logique séquentielle

Bascules D asynchrone et synchrone avec blocs gardés

```
entity BDA is
        port (      d, raz, clk : in std_logic;
                    Q : out std_logic);
end BDA;

architecture bloc of BDA is
begin
  B : block ((clk ’event and clk = ‘1’)
                    or raz =‘0’)
    begin
        Q <= guarded ‘0’ when raz = ‘0’ ;
            else d when clk = ‘1’ ;
            else Q;
    end block B;
end bloc;
```

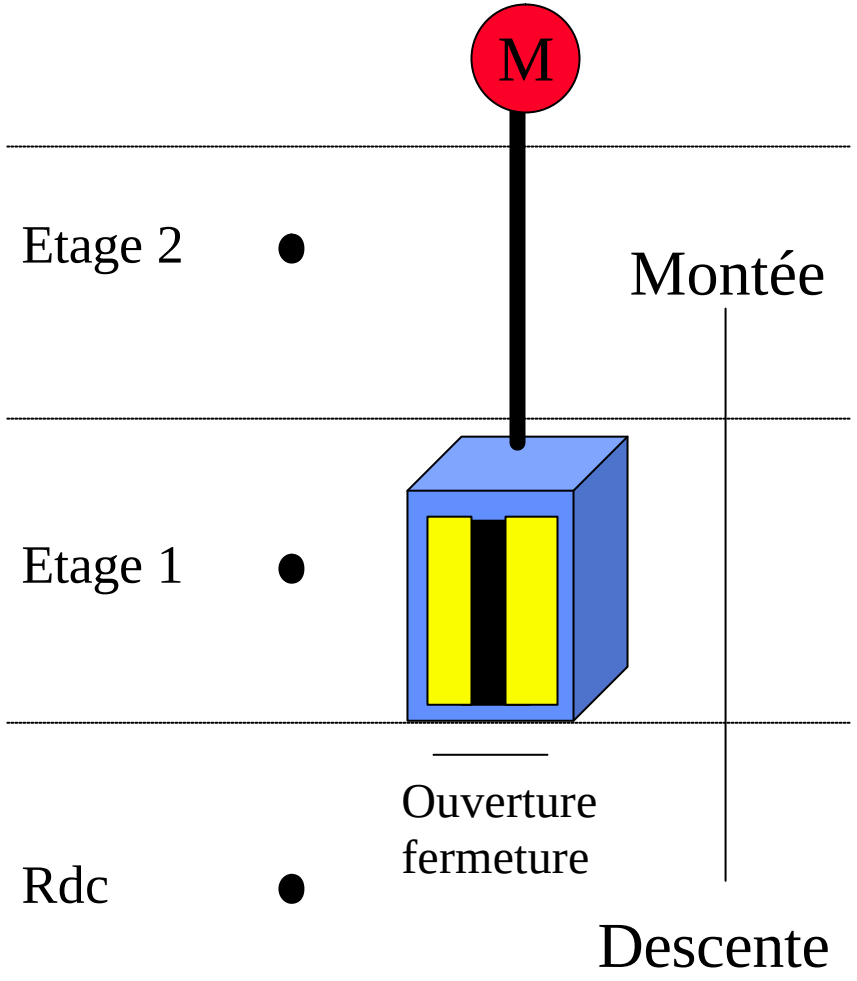
Asynchrone

```
entity BDS is
        port (      d, raz, clk : in std_logic;
                    Q : out std_logic);
end BDS;

architecture bloc of BDS is
begin
  B : block (clk = ‘1’ and clk’event)
    begin
        Q <= guarded ‘0’ when raz = ‘0’ ;
            else d when clk = ‘1’ ;
            else  Q;
    end block B;
end bloc;
```

Synchrone

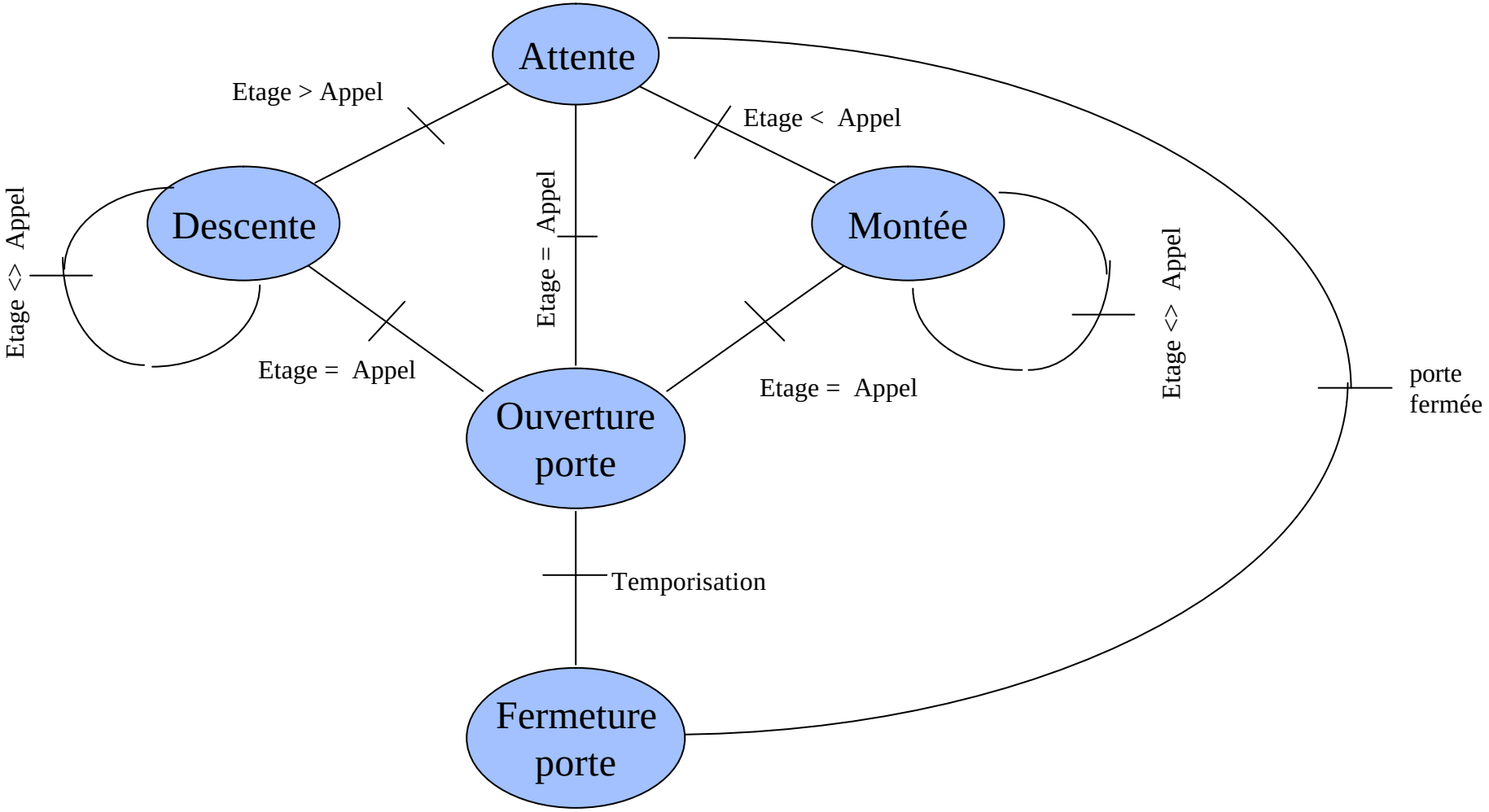
10) Logique séquentielle

- Réalisation du contrôleur d'un ascenseur :
  - il s'agit de commander :
    - ✓ le moteur d'ouverture et de fermeture des portes
    - ✓ le moteur de monter et de descente de l'ascenseur
    - ✓ d'assurer une temporisation entre l'ouverture et la fermeture des portes
  - on dispose :
    - ✓ d'une information indiquant, à tout moment, l'étage courant
    - ✓ de capteurs d'appel de l'ascenseur
    - ✓ de capteur de porte fermée

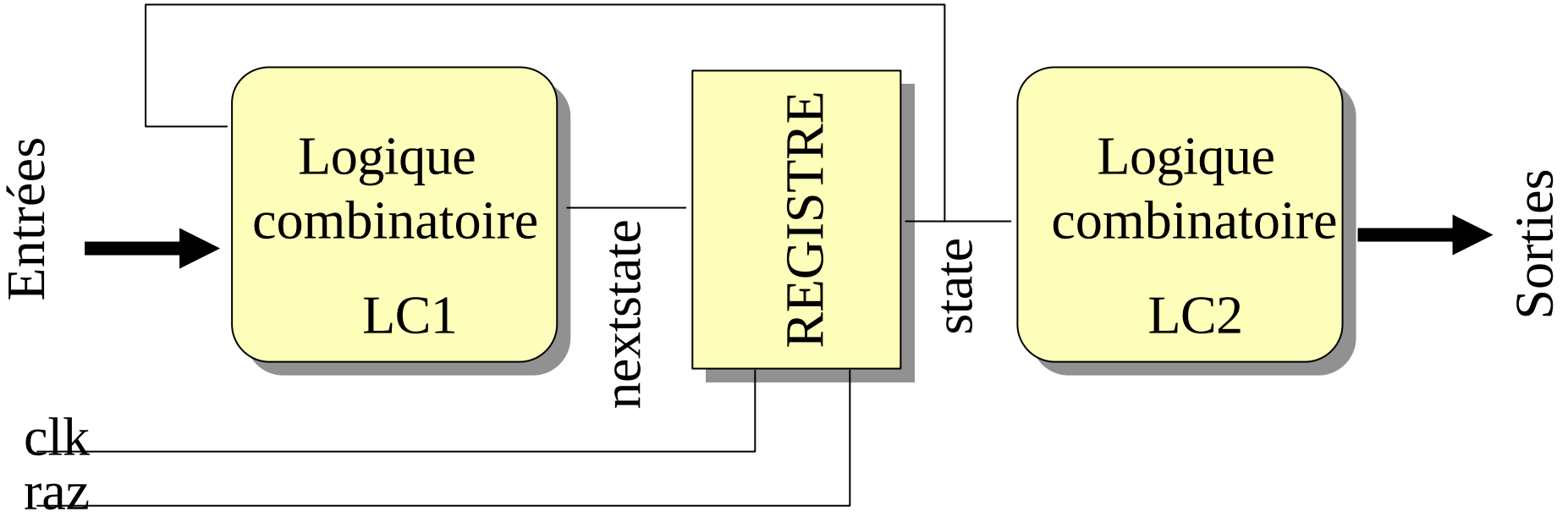


10) Logique séquentielle

Matérialisation d'une machine d'états



10) Logique séquentielle



Modélisations envisageables :

1) 1processus : **(LC1 + REGISTRE + LC2)**
2) 2 processus : **(LC1 + REGISTRE) et LC2**
   ou **LC1 et (REGISTRE + LC2)**
3) 3 processus : **LC1, REGISTRE et LC2**

Vue externe

```
entity Controleur is
        port (  Appel, Etage : in integer ;
                Open, Close, clk, raz : in bit ;
                Monter, Descendre, Ouvrir, Fermer : out bit );
end Controleur ;
```

10) Logique séquentielle

Vue interne

```
architecture comportement of Ascenseur is
        type etat is (Attente, Descente, Montee, Ouverture, Fermeture);
        signal state : etat := Attente;
        signal nextstate : etat;
        signal Etage : integer := 0;
begin
        LC1 : process
                begin                   -- calcul l'etat suivant en fonction des entrees
                ...                     -- et de l'état courant
                wait on Etage, Appel, Open, Close, state;
                end process LC1 ;
        REGISTRE : process
                begin                   -- change d'état au front montant de l'horloge
                …
                wait on clk, raz;
                end process REGISTRE ;
        LC2 : process
                begin                   -- calcul l'etat des sorties en fonction de l'etat
                ...                     -- courant
                wait on state;
                end process LC2 ;
end comportement;
```

10) Logique séquentielle


```
LC1 : process
      begin
     case state is
            when Attente =>
                  if Etage > Appel then          nextstate <= Descente;
                  elsif Etage < Appel then       nextstate <= Montee;
                  else                           nextstate <= Ouverture;
                  end if ;
            when Descente =>
                  if Etage = Appel then          nextstate <= Ouverture ;
                  else                           nextstate <= Descente ;
                  end if ;
            when Montee =>
                  if Etage = Appel then          nextstate <= Ouverture ;
                  else                           nextstate <= Montee ;
                  end if ;
            when Ouverture =>
                  if  Open = '1' then            nextstate <= Fermeture ;
                  else                           nextstate <= Ouverture ;
                  end if ;
            when Fermeture  =>
                  if Close = '1' then            nextstate  <= Attente ;
                  else                           nextstate  <= Fermeture ;
                  endif ;
      end case ;
      wait on Etage, Appel, Open,  Close, state;
end process LC1 ;
```

10) Logique séquentielle

```
REGISTRE : process
begin
        if raz = '0' then
                state <= Attente ;
        elsif clk'event and clk = '1' then
                state <= nextstate ;
        end if;
        wait on clk, raz;
end process REGISTRE ;
```

```
LC2 : process
        variable m, d, o, f : bit;
begin
        case state is
                when Attente =>
                        m := '0'; d := '0'; o := '0'; f := '0' ;
                when Montee =>
                        m := '1'; d := '0'; o := '0'; f := '0' ;
                when Descente =>
                        m := '0'; d := '1'; o := '0'; f := '0' ;
                when Ouverture =>
                        m := '0'; d := '0'; o := '1'; f := '0' ;
                when Fermeture =>
                        m := '0'; d := '0'; o := '0'; f := '1' ;
        end case ;
        Monter      <= m;
        Descendre <= d;
        Ouvrir      <= o;
        Fermer      <= f;
        wait on state;
end process LC2 ;
```

Packages

11) Paquetages standards

- Paquetages standards : 2 paquetages :
  - standard :
    - ✓ rassemble les déclarations de types, de sous types, de fonctions extrêmement utiles (voir indispensable)
  - textio :
    - ✓ fournit les primitives d'entrées sorties ASCII de VHDL
    - ✓ utilisé pour le dialogue avec une console (debuggage par exemple)
    - ✓ Ce paquetage est un raté de VHDL tout ne répond pas à la grammaire de vhdl
    - ✓ Conséquences :
      - chaque constructeur de compilateur a proposé son paquetage textio
      - donc plus vraiment un standard

11) Paquetages standards

Package standard : Définition

- package standard is
  - type BOOLEAN is (FALSE, TRUE);
  - type BIT is (‘0’, ‘1’);
  - type CHARACTER is (NUL, ..., ‘0’,’1’,’2’, ... ,’a’, .., ‘z’)
  - type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);
  - type INTEGER is range -2 147 483 648 to 2 147 483 647;
  - type real is range -16. 10(32) to 16. 10(32);
  - type TIME is range -2(64) to 2(64)

    units fs

    ps = 1000 fs;

    ns = 1000 ps

    ms = 1000 ns

11) Paquetages standards

- function Now return time ;
- 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 ;

- end standard ;

- chaîne : string (1 to N) ;
- V : bit_vector (15 downto 0) ;

11) Paquetages standards

Package Textio : Définition

- package textio is
  - type LINE is access string ; -- pointeur sur une ligne
  - type text is file of string ; -- fichier de caracteres
  - type SIDE is (right, left); -- justifier à gauche ou à droite
  - subtype width is natural ; -- C'est un simple renommage
    -- utilisé pour les longueurs de chaines
  - file input : text is in "std_input" ; -- clavier
  - file output : text is out "std_out" ; -- écran
  - procedure readline (variable F : in text ; L : out line) ;
  - procedure writeline (variable F : out text ; L : in line);

11) Paquetages standards

- 16 procédures read :
  - ✓ extraction de la valeur du type demandé dans la ligne (LINE)
  - ✓ ces procédures rendent la suite de la ligne
- 8 procédures write :
  - ✓ permettent de construire la ligne (dans une variable LINE) avant de l'écrire
- function endline (L : in Line) return Boolean ;
- function endfile (F : in text) return Boolean ;

❑ end textio ;

Belles surcharges !!

11) Paquetages IEEE

- Les paquetages normalisés :
  - package std_logic_1164 :
    - ✓ définition des types :
      - std_logic
      - std_logic_vector
    - ✓ définition des fonctions logiques de base sur ces types :
      - AND, OR, NOR, XOR, etc
  - package std_logic_arith :
    - ✓ surcharge des opérateurs de base :
      - +, -, *,
    - ✓ surcharge des comparateurs de base :
      - <=, >=, /=, =

11) Paquetages IEEE

- package std_logic_signed :
  - ✓ surcharge des opérateurs de base :
    - +, -, *,
  - ✓ surcharge des comparateurs de base :
    - <=, >=, /=, =
  - ✓ définition d'une fonction de conversion :
    - std_logic_vector ====> Integer
    - prise en compte du signe

- package std_logic_unsigned :
  - ✓ surcharge des opérateurs de base :
    - +, -, *,
  - ✓ surcharge des comparateurs de base :
    - <=, >=, /=, =
  - ✓ définition d'une fonction de conversion :
    - std_logic_vector ====> Integer
    - prise en compte du signe

11) Paquetages IEEE

- Attention :
  - ces paquetages sont d'une utilisation délicate :
  - on ne peut utiliser les 2 paquetages signed ey unsigned simultanément
  - les fonctions de conversion renvoient des valeurs différentes (fonction du bit de signe)
  - tous les opérateurs de base ne sont pas surchargés
  - etc etc

11) Paquetages IEEE

```
-- ------------------------------------------------------------------
--
--  Title     :  std_logic_1164 multi-value logic system
--  Library   :  This package shall be compiled into a library
--            :  symbolically named IEEE.
--            :
--  Developers:  IEEE model standards group (par 1164)
--  Purpose   :  This packages defines a standard for designers
--            :  to use in describing the interconnection data types
--            :  used in vhdl modeling.
--            :
--  Limitation:  The logic system defined in this package may
--            :  be insufficient for modeling switched transistors,
--            :  since such a requirement is out of the scope of this
--            :  effort. Furthermore, mathematics, primitives,
--            :  timing standards, etc. are considered orthogonal
--            :  issues as it relates to this package and are therefore
--            :  beyond the scope of this effort.
--            :
--  Note      :  No declarations or definitions shall be included in,
--            :  or excluded from this package. The "package declaration"
--            :  defines the types, subtypes and declarations of
--            :  std_logic_1164. The std_logic_1164 package body shall be
--            :  considered the formal definition of the semantics of
--            :  this package. Tool developers may choose to implement
--            :  the package body in the most efficient manner available
--            :  to them.
--            :
-- ------------------------------------------------------------------
--  modification history:
-- ------------------------------------------------------------------
--  version | mod. date:|
--   v4.200 | 01/02/92  |
-- ------------------------------------------------------------------

PACKAGE std_logic_1164 IS

  ------------------------------------------------------------------
  -- logic state system  (unresolved)
  ------------------------------------------------------------------

  TYPE std_ulogic IS ( 'U',  -- Uninitialized
                       'X',  -- Forcing  Unknown
                       '0',  -- Forcing  0
                       '1',  -- Forcing  1
                       'Z',  -- High Impedance
                       'W',  -- Weak     Unknown
                       'L',  -- Weak     0
                       'H',  -- Weak     1
                       '-'   -- Don't care
                     );
  ------------------------------------------------------------------
  -- unconstrained array of std_ulogic for use with the resolution function
  ------------------------------------------------------------------
  TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;

  ------------------------------------------------------------------
  -- resolution function
  ------------------------------------------------------------------
  FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;

  ------------------------------------------------------------------
  -- *** industry standard logic type ***
  ------------------------------------------------------------------
  SUBTYPE std_logic IS resolved std_ulogic;

  ------------------------------------------------------------------
  -- unconstrained array of std_logic for use in declaring signal arrays
  ------------------------------------------------------------------
  TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;

  ------------------------------------------------------------------
  -- common subtypes
  ------------------------------------------------------------------
  SUBTYPE X01     IS resolved std_ulogic RANGE 'X' TO '1'; -- ('X','0','1')
  SUBTYPE X01Z    IS resolved std_ulogic RANGE 'X' TO 'Z'; -- ('X','0','1','Z')
  SUBTYPE UX01    IS resolved std_ulogic RANGE 'U' TO '1'; -- ('U','X','0','1')
  SUBTYPE UX01Z   IS resolved std_ulogic RANGE 'U' TO 'Z'; --
('U','X','0','1','Z')

  ------------------------------------------------------------------
  -- overloaded logical operators
  ------------------------------------------------------------------
  FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
  FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
```

11) Paquetages IEEE

```
FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
function "xnor" ( l : std_ulogic; r : std_ulogic ) return ux01;
FUNCTION "not"  ( l : std_ulogic                 ) RETURN UX01;

-------------------------------------------------------------------
-- vectorized overloaded logical operators
-------------------------------------------------------------------
FUNCTION "and"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "and"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "nand" ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "nand" ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "or"   ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "or"   ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "nor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "nor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "xor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "xor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

function "xnor" ( l, r : std_logic_vector  ) return std_logic_vector;
function "xnor" ( l, r : std_ulogic_vector ) return std_ulogic_vector;

FUNCTION "not"  ( l : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "not"  ( l : std_ulogic_vector ) RETURN std_ulogic_vector;

-------------------------------------------------------------------
-- conversion functions
-------------------------------------------------------------------
FUNCTION To_bit      ( s : std_ulogic;       xmap : BIT := '0') RETURN BIT;
FUNCTION To_bitvector ( s : std_logic_vector ; xmap : BIT := '0') RETURN BIT_VECTOR;
FUNCTION To_bitvector ( s : std_ulogic_vector; xmap : BIT := '0') RETURN BIT_VECTOR;
FUNCTION To_StdULogic      ( b : BIT             ) RETURN std_ulogic;
FUNCTION To_StdLogicVector  ( b : BIT_VECTOR       ) RETURN std_logic_vector;
FUNCTION To_StdLogicVector  ( s : std_ulogic_vector ) RETURN std_logic_vector;
FUNCTION To_StdULogicVector ( b : BIT_VECTOR       ) RETURN std_ulogic_vector;
FUNCTION To_StdULogicVector ( s : std_logic_vector  ) RETURN std_ulogic_vector;

-------------------------------------------------------------------
  -- strength strippers and type convertors
  -------------------------------------------------------------------
  FUNCTION To_X01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
  FUNCTION To_X01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
  FUNCTION To_X01  ( s : std_ulogic       ) RETURN  X01;
  FUNCTION To_X01  ( b : BIT_VECTOR       ) RETURN  std_logic_vector;
  FUNCTION To_X01  ( b : BIT_VECTOR       ) RETURN  std_ulogic_vector;
  FUNCTION To_X01  ( b : BIT              ) RETURN  X01;

  FUNCTION To_X01Z ( s : std_logic_vector  ) RETURN  std_logic_vector;
  FUNCTION To_X01Z ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
  FUNCTION To_X01Z ( s : std_ulogic       ) RETURN  X01Z;
  FUNCTION To_X01Z ( b : BIT_VECTOR       ) RETURN  std_logic_vector;
  FUNCTION To_X01Z ( b : BIT_VECTOR       ) RETURN  std_ulogic_vector;
  FUNCTION To_X01Z ( b : BIT              ) RETURN  X01Z;

  FUNCTION To_UX01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
  FUNCTION To_UX01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
  FUNCTION To_UX01  ( s : std_ulogic       ) RETURN  UX01;
  FUNCTION To_UX01  ( b : BIT_VECTOR       ) RETURN  std_logic_vector;
  FUNCTION To_UX01  ( b : BIT_VECTOR       ) RETURN  std_ulogic_vector;
  FUNCTION To_UX01  ( b : BIT              ) RETURN  UX01;

  -------------------------------------------------------------------
  -- edge detection
  -------------------------------------------------------------------
  FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN;
  FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN;

  -------------------------------------------------------------------
  -- object contains an unknown
  -------------------------------------------------------------------
  FUNCTION Is_X ( s : std_ulogic_vector ) RETURN  BOOLEAN;
  FUNCTION Is_X ( s : std_logic_vector  ) RETURN  BOOLEAN;

  FUNCTION Is_X ( s : std_ulogic       ) RETURN  BOOLEAN;

END std_logic_1164;
```

11) Paquetages IEEE


```
------------------------------------------------------------------------
--                                                                    --
-- Copyright (c) 1990,1991,1992 by Synopsys, Inc.  All rights reserved. --
--                                                                    --
-- This source file may be used and distributed without restriction    --
-- provided that this copyright statement is not removed from the file --
-- and that any derivative work contains this copyright notice.       --
--                                                                    --
--   Package name: STD_LOGIC_ARITH                                    --
--                                                                    --
--   Purpose:                                                         --
--    A set of arithemtic, conversion, and comparison functions       --
--    for SIGNED, UNSIGNED, SMALL_INT, INTEGER,                       --
--    STD_ULOGIC, STD_LOGIC, and STD_LOGIC_VECTOR.                    --
--                                                                    --
------------------------------------------------------------------------

-------------------------------------------------
-- Attributes added to invoke MTI builtin functions
-------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

package std_logic_arith is

    type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
    type SIGNED is array (NATURAL range <>) of STD_LOGIC;
    subtype SMALL_INT is INTEGER range 0 to 1;

    attribute builtin_subprogram : string;

    ----------------
    -- add operators
    ----------------
    function "+"(L: UNSIGNED; R: UNSIGNED) return UNSIGNED;
    attribute builtin_subprogram of
        "+"[UNSIGNED, UNSIGNED return UNSIGNED]: function is "stdarith_plus_uuu";

    function "+"(L: SIGNED; R: SIGNED) return SIGNED;
    attribute builtin_subprogram of
        "+"[SIGNED, SIGNED return SIGNED]: function is "stdarith_plus_sss";

    function "+"(L: UNSIGNED; R: SIGNED) return SIGNED;
    attribute builtin_subprogram of
        "+"[UNSIGNED, SIGNED return SIGNED]: function is "stdarith_plus_uss";

    function "+"(L: SIGNED; R: UNSIGNED) return SIGNED;
    attribute builtin_subprogram of
        "+"[SIGNED, UNSIGNED return SIGNED]: function is "stdarith_plus_sus";

    function "+"(L: UNSIGNED; R: INTEGER) return UNSIGNED;
    attribute builtin_subprogram of
        "+"[UNSIGNED, INTEGER return UNSIGNED]: function is "stdarith_plus_uiu";

    function "+"(L: INTEGER; R: UNSIGNED) return UNSIGNED;
    attribute builtin_subprogram of
        "+"[INTEGER, UNSIGNED return UNSIGNED]: function is "stdarith_plus_iuu";

    function "+"(L: SIGNED; R: INTEGER) return SIGNED;
    attribute builtin_subprogram of
        "+"[SIGNED, INTEGER return SIGNED]: function is "stdarith_plus_sis";

    function "+"(L: INTEGER; R: SIGNED) return SIGNED;
    attribute builtin_subprogram of
        "+"[INTEGER, SIGNED return SIGNED]: function is "stdarith_plus_iss";

    function "+"(L: UNSIGNED; R: STD_ULOGIC) return UNSIGNED;
    attribute builtin_subprogram of
        "+"[UNSIGNED, STD_ULOGIC return UNSIGNED]: function is "stdarith_plus_uxu";

    function "+"(L: STD_ULOGIC; R: UNSIGNED) return UNSIGNED;
    attribute builtin_subprogram of
        "+"[STD_ULOGIC, UNSIGNED return UNSIGNED]: function is "stdarith_plus_xuu";

    function "+"(L: SIGNED; R: STD_ULOGIC) return SIGNED;
    attribute builtin_subprogram of
        "+"[SIGNED, STD_ULOGIC return SIGNED]: function is "stdarith_plus_sxs";

    function "+"(L: STD_ULOGIC; R: SIGNED) return SIGNED;
    attribute builtin_subprogram of
        "+"[STD_ULOGIC, SIGNED return SIGNED]: function is "stdarith_plus_xss";
```

11) Paquetages IEEE


```
function "+"(L: UNSIGNED; R: UNSIGNED) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[UNSIGNED, UNSIGNED return STD_LOGIC_VECTOR]: function is "stdarith_plus_uuu";

  function "+"(L: SIGNED; R: SIGNED) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[SIGNED, SIGNED return STD_LOGIC_VECTOR]: function is "stdarith_plus_sss";

  function "+"(L: UNSIGNED; R: SIGNED) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[UNSIGNED, SIGNED return STD_LOGIC_VECTOR]: function is "stdarith_plus_uss";

  function "+"(L: SIGNED; R: UNSIGNED) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[SIGNED, UNSIGNED return STD_LOGIC_VECTOR]: function is "stdarith_plus_sus";

  function "+"(L: UNSIGNED; R: INTEGER) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[UNSIGNED, INTEGER return STD_LOGIC_VECTOR]: function is "stdarith_plus_uiu";

  function "+"(L: INTEGER; R: UNSIGNED) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[INTEGER, UNSIGNED return STD_LOGIC_VECTOR]: function is "stdarith_plus_iuu";

  function "+"(L: SIGNED; R: INTEGER) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[SIGNED, INTEGER return STD_LOGIC_VECTOR]: function is "stdarith_plus_sis";

  function "+"(L: INTEGER; R: SIGNED) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[INTEGER, SIGNED return STD_LOGIC_VECTOR]: function is "stdarith_plus_iss";

  function "+"(L: UNSIGNED; R: STD_ULOGIC) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[UNSIGNED, STD_ULOGIC return STD_LOGIC_VECTOR]: function is "stdarith_plus_uxu";

  function "+"(L: STD_ULOGIC; R: UNSIGNED) return STD_LOGIC_VECTOR;
  attribute builtin_subprogram of
    "+"[STD_ULOGIC, UNSIGNED return STD_LOGIC_VECTOR]: function is "stdarith_plus_xuu";
```