Séminaire: Introduction au VHDL I

Sébastien Roy, Réjean Fontaine et Daniel Dalle
courriel: Sebastien.Roy13@usherbrooke.ca
©Tous droits réservés 2018-2024
Département de génie électrique et de génie informatique

Université de Sherbrooke

8 janvier 2024

UDS
Université de Sherbrooke

Plan

Section 1

Systèmes numériques

Qu'est-ce qu'un système numérique ?

Définition

Un système numérique, à la base, est un assemblage de plusieurs "boîtes" ou *modules logiques*, interconnectés de manière à réaliser une fonction utile.

Modules logiques, définition

Définition

Un module logique est une entité comprenant une ou plusieurs entrée(s), une ou plusieurs sortie(s) et dont les sorties sont obtenues en effectuant une ou plusieurs opérations logiques sur les entrées.

- Cette définition est générale...
- Elle ne fait aucunement référence à la réalisation physique du module.
- Un module logique est donc une abstraction...
- ...pouvant correspondre ou non à une réalisation en logiciel ou en matériel dans un format quelconque.

UDS Université de Sherbrooke

Fonctions logiques

Définition

Une fonction logique est un module logique sans mémoire, c-à-d

1. tout changement dans les entrées est immédiatement reflété à la sortie ;
2. à tout instant, la sortie est uniquement fonction des entrées actuelles.

Exemples de fonctions logiques

Exemple 1 : $y = \log(x)$

- entrée continue, sortie continue
- tout changement dans x se répercute immédiatement sur y

Exemple 2 : une porte logique idéale

Délai entrée-sortie nul : porte ET, OU, NON, NON-ET, NON-OU, OUX, etc.

Les fonctions numériques, comme à l'ex. 1, supposent des représentations de quantités numériques par des vecteurs de bits.

UDS
Université de Sherbrooke

Processus logique

Définition

Un processus logique est un module logique avec mémoire, c-à-d

- La sortie dépend non seulement des entrées actuelles mais également des entrées passées

Exemple : Processus modal ou automate fini [a]

a. Aussi machine à états ou machine à états finie ou automate à nombre d'états fini.

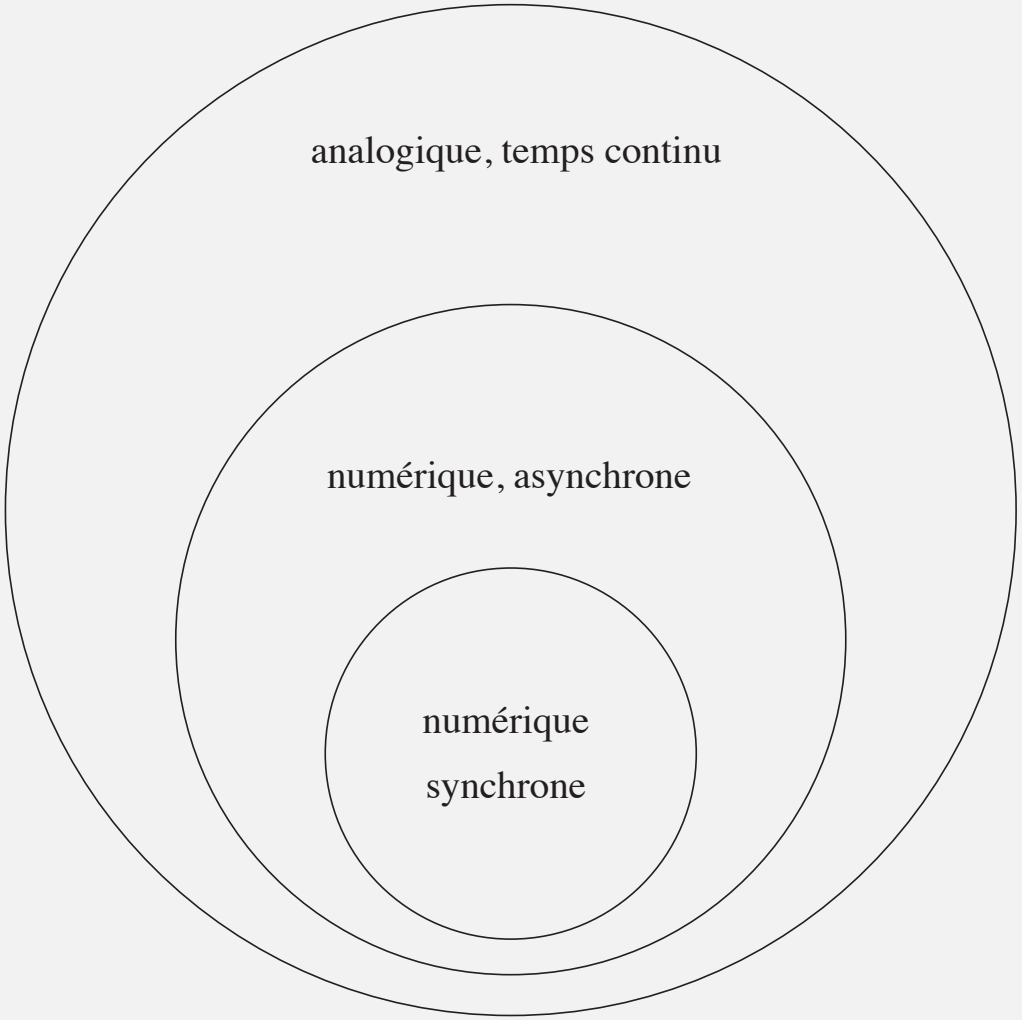
- les sorties dépendent non seulement des entrées mais également de l'état courant / mode courant ;
- certaines combinaisons d'entrées peuvent modifier l'état / le mode, ce qui altère le comportement subséquent du processus.

Modules logiques

- Les modules logiques sont donc les “briques de base” de tout circuit ou système numérique.
- Un module logique plus complexe est constitué de modules logiques plus simples : hiérarchie.
- En conception, il est très important de distinguer clairement les deux types de module :
  - les fonctions logiques (sans mémoire, logique combinatoire) → APP 1
  - les processus logiques (automate fini, avec mémoire, logique séquentielle) → APP 2

UDS Université de Sherbrooke

Types de circuits



- À la base, tous les circuits sont analogiques et en temps continu.
- On peut structurer les circuits analogiques en leur imposant certains formalismes :
  - Discrétisation des niveaux
  - Discrétisation du temps

UdS
Université de Sherbrooke

La discrétisation en temps et en niveau

1. Circuits analogiques : niveaux de voltage continus, temps continu
2. Circuits logiques asynchrones : niveaux de voltage discrets, temps continu
   - Avantages : précision, tolérance au bruit, représentation de quantités discrètes
3. Circuits logiques synchrones : niveaux de voltage discrets, temps discret
   - Avantages : design plus simple, évite les erreurs de conception, fonctionnement prévisible et déterministe

UDS Université de Sherbrooke

Section 2

VHDL et modélisation

Qu'est-ce que VHDL ?

- A- Very Hard Description Language ?
- B- VHSIC* Hardware Description Language
- * VHSIC : Very High Speed Integrated Circuit
- VHDL est un langage permettant la construction de modèles formels de systèmes numériques.

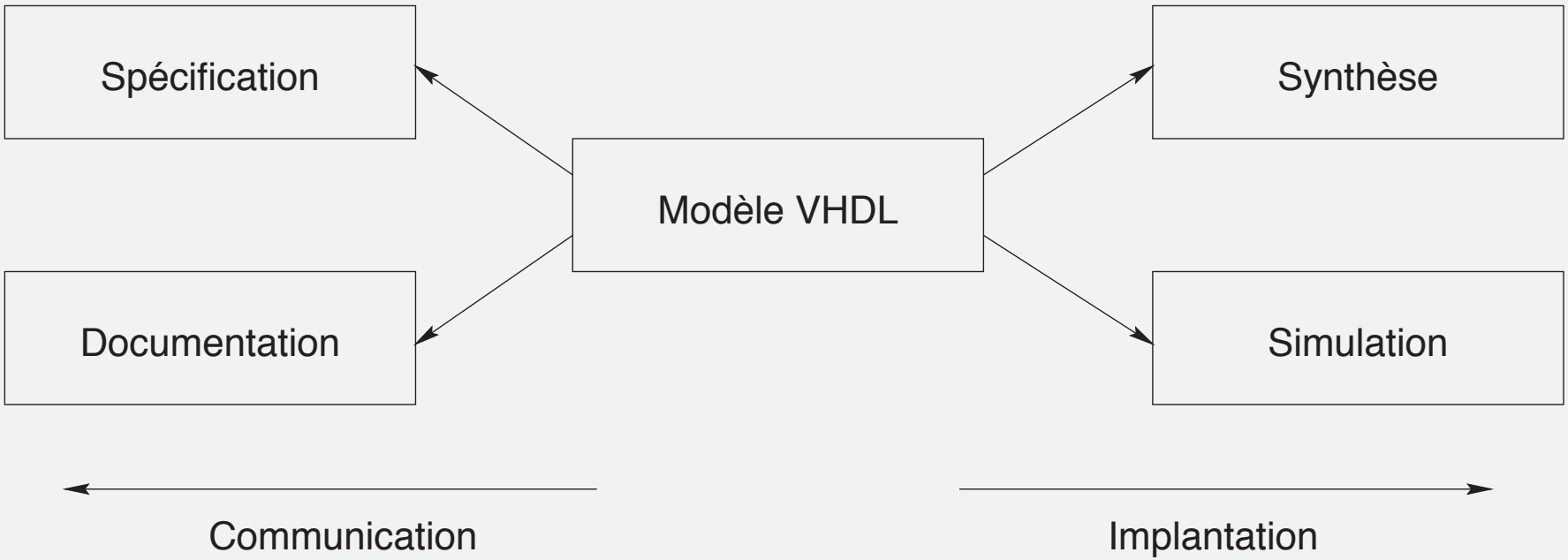
La modélisation de systèmes numériques

A quoi sert la modélisation ?

1. spécification
2. documentation
3. test et simulation
4. vérification formelle
5. synthèse

UDS Université de Sherbrooke

La modélisation de systèmes numériques — 2



Université de Sherbrooke

Applications de la modélisation

A quoi sert la modélisation dans le processus de conception ?

- supporte le travail en équipes ;
- réduit le temps et le coût de conception ;
- rend le processus de modélisation plus fiable ;
- évite les erreurs de design ;
- facilite la gestion de projets à grande échelle ;
- permet la réutilisation des modèles pour différentes technologies-cibles.

UDS
Université de Sherbrooke

Les origines

- En 1981, le DoD des E.-U. a formulé le besoin d'un système de description formel et standard des circuits dans le cadre de son programme VHSIC.
- A cette époque, les fournisseurs du DoD avait chacun leur HDL privé ce qui limitait l'échange des designs.
- Le but premier est de décrire de manière formelle et sans ambiguïté le fonctionnement du système.
- Ceci devait remplacer les volumineux et rébarbatifs manuels techniques.
- On sentait le besoin de décrire les designs sous une forme indépendante de la technologie-cible.
- Du même coup, de telles descriptions seraient moins vulnérables à la rapide désuétude technologique.

Bornes historiques

- 81 - le DoD des E.-U. amorce une étude afin de régler le problème du court cycle de vie des circuits électroniques
- 83-85 - développement du langage sous contrat par Intermetrics/IBM/TI
- 86 - les droits sur VHDL sont transférés à l'IEEE
- 87 - publication du standard IEEE 1076-87 (VHDL 87)
- 87 - le standard MIL STD 454 oblige les fabricants à fournir une description VHDL avec tous les ASIC développés pour l'armée américaine
- 94 - le standard révisé est publié (VHDL 93 ou IEEE 1076-93).
- 2001 - révision mineure (VHDL 2001 ou IEEE 1076-2001)
- 2006 - troisième version (VHDL 2006 ou IEEE 1076-2006) ; incorpore les standards dérivés
- janvier 2009 - 4e version (VHDL 4.0 ou IEEE 1076-2008)

UDS Université de Sherbrooke

Qu'est-ce que le VHDL ?

- Le VHDL n'est pas un langage de programmation
- C'est plutôt
  - un langage de modélisation ;
  - un outil devenu incontournable à cause de la complexité grandissante des systèmes numériques (on se perdait dans les schémas à pages multiples) ;
  - une façon de décrire formellement et rigoureusement le comportement et la structure d'un circuit.

UDS Université de Sherbrooke

Séquentiel vs parallèle

Langages de programmation classiques (C, C++, Python, Java, etc.)

- Paradigme von Neumann : séquence successive d'opérations
- Des mécanismes comme les *fils* (threads) permettent un certain parallélisme, mais séquentiel à la base
- Les instructions sont exécutées en séquence, une après l'autre.
- Chaque instruction a donc un effect ponctuel dans le temps. Elle se termine et l'exécution passe à la suivante.
- Le lien de causalité dépend de la séquence d'exécution : si une instruction affecte une variable, l'instruction suivante verra la nouvelle valeur.

VHDL

- Les systèmes numériques sont naturellement parallèles — il se passe plusieurs choses en même temps.
- VHDL procure des contextes parallèles et séquentiels.
- Le contexte naturel est parallèle (corps d'une architecture).
- Les contextes séquentiels se rapprochent du paradigme von Neumann, mais avec des nuances.
- Les commandes VHDL se divisent nettement en deux catégories, séquentielles et parallèles.
- Les commandes parallèles ont un effet continu dans le temps, sans début ni fin.
- L'ordre des commandes parallèles n'a pas d'importance.
- Le lien de causalité dépend de l'ordre des évènements.

UDS
Université de Sherbrooke

Section 3

Concepts de base en VHDL

Une description VHDL

Définition

Une description VHDL ou unité de design est composée d'une **entête**, d'une **entité** et d'une **architecture**.

Description générique

$$
\begin{array}{rcl}
\textit{entête} & \rightarrow & \left\{ \begin{array}{l} \textbf{library}\ \text{ieee}\,; \\ \textbf{use}\ \text{ieee.std\_logic\_1164.}\textbf{all}\,; \end{array} \right. \\
\textit{entité} & \rightarrow & \left\{ \begin{array}{l} \textbf{entity}\ \ll \textit{nom\_entité} \gg\ \textbf{is} \\ \quad \ll \textit{ports} \gg \\ \textbf{end}\ \ll \textit{nom\_entité} \gg\,; \end{array} \right. \\
\textit{architecture} & \rightarrow & \left\{ \begin{array}{l} \textbf{architecture}\ \ll \textit{nom\_arch} \gg\ \textbf{of}\ \ll \textit{nom\_entité} \gg\ \textbf{is} \\ \quad \ll \textit{déclarations} \gg \\ \textbf{begin} \\ \quad \ll \textit{corps} - \text{zone parallèle} \gg \\ \textbf{end architecture}\ \ll \textit{nom\_arch} \gg\,; \end{array} \right.
\end{array}
$$

Entête

- Un peu comme les directives **#include** en C
- Dans l'entête, on rend visible des bibliothèques (commande **library**).
- Dans une bibliothèque, on peut appeler un paquet (commande **use**).
- Pour rendre toutes les définitions d'un paquet visible, on écrit : **use** ≪*bibliothèque*≫.≪*paquet*≫.**all** ;
- Dans la pratique courante, l'entête est pratiquement toujours la même.

Entête standard

```
library ieee ;
use ieee.std_logic_1164.all ;
```

Université de Sherbrooke

Entité

Entité générique

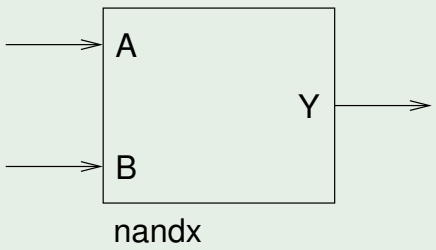
entity ≪*nom_entité*≫ **is**
≪ *ports* ≫
end ≪*nom_entité*≫ ;

Définition

L'entité décrit les ports du "bloc", c-à-d son interface avec le monde extérieur.

Exemple d'entité

```
entity nandx is
    port(
        A, B : in std_logic;
        Y : out std_logic
    );
end nandx;
```



Architecture

Architecture générique

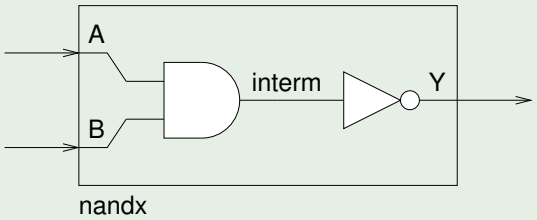
architecture ≪*nom_arch*≫ **of** ≪*nom_entité*≫ **is**
    ≪*déclarations*≫
begin
    ≪*corps* – zone parallèle≫
end architecture ≪*nom_arch*≫ ;

Définition

L'architecture décrit le fonctionnement interne du bloc.

Exemple d'architecture

```
architecture flot of nandx is
    signal interm : std_logic;
begin
    interm <= a and b;
    y <= not interm;
end architecture flot;
```



- Les deux commandes ici sont des assignation de signaux parallèles.

Université de Sherbrooke

Les processus

Définition

Le processus est une commande parallèle qui permet d'inclure dans une architecture une description de type séquentiel, c-à-d elle fournit un contexte d'exécution séquentiel. L'exécution est déclenchée par un évènement sur un des signaux dans la liste de sensibilité.

Structure d'un processus

≪étiquette≫ : **process**(*≪liste de sensibilité≫*) **is**
    ≪ déclarations locales ≫ ;
begin
    ≪commandes séquentielles≫ ;
end process *≪étiquette≫* ;

Processus — logique combinatoire

- Un processus peut servir à décrire de la logique combinatoire.
- On inclut alors tous les signaux d'entrée dans la liste de sensibilité.
- Les signaux d'entrée sont tous les signaux qu'on retrouve dans le membre de droite d'assignations dans le corps du processus.

Structure d'un processus combinatoire

≪étiquette≫ : **process**(*≪entrées≫*) **is**
  ≪ déclarations locales ≫ ;
begin
  ≪commandes séquentielles≫ ;
end process *≪étiquette≫* ;

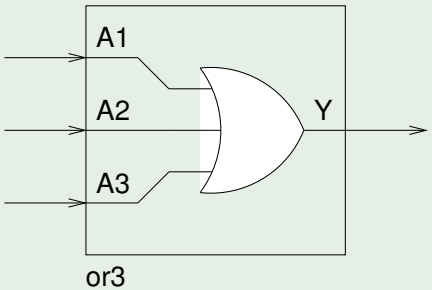
Assignation de base

- Tous les types d'assignation parallèle ont un processus équivalent.

Exemple

Assignation

```
architecture flot of or3 is
begin
    Y <= A1 or A2 or A3 ;
end architecture flot ;
```



Processus équivalent

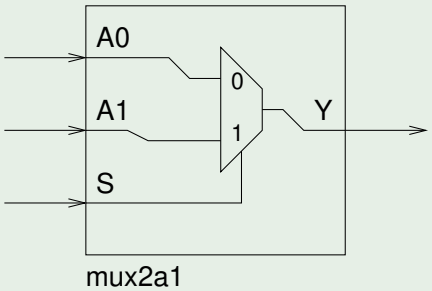
```
architecture flot of or3 is
begin
    P_OU : process( A1, A2, A3 ) is
    begin
        Y <= A1 or A2 or A3 ;
    end process P_OU ;
end architecture flot ;
```

Assignation conditionnelle

Exemple

Assignation

```
architecture flot of mux2a1 is
begin
    Y <= A0 when S='0' else A1;
end architecture flot;
```



mux2a1

Processus équivalent

```
architecture flot of mux2a1 is
begin
    muxpr : process( A0, A1, S ) is
    begin
        if S='0' then
            Y <= A0;
        else
            Y <= A1;
        end if;
    end process muxpr;
end architecture flot;
```

Université de Sherbrooke

Commande **if... then...else**

Commande séquentielle

if ≪*condition*≫ **then**
    ≪*commandes séquentielles*≫
{**elseif** ≪*condition*≫ **then**
    ≪*commandes séquentielles*≫}
[**else**
    ≪*commandes séquentielles*≫]
end if ;

Université de Sherbrooke

Bits et vecteurs de bits — 1

- Le VHDL standard (IEEE 1076) définit les types **bit** et **bit_vector**.
- Vecteur de bits → série de bits, indexée avec une plage ascendante ou descendante.
- Plage descendante : étant donné
  signal x : bit_vector(3 **downto** 0) := "0010";
  on observe que x(1) vaut 1. Ici, l'index correspond au poids de la position.
- Plage ascendante : étant donné
  signal x : bit_vector(0 **to** 3) := "0010";
  on observe que c'est maintenant x(2) qui vaut 1.
- Étant donné
  signal x : bit_vector(7 **downto** 4) := "0100";
  on observe ici que c'est x(6) qui vaut 1.
- Note : les plages descendantes sont privilégiées dans le code qui vous est fourni dans les APP 1 et 2. Il est conseillé de se conformer à cette convention dans votre code.

UDS
Université de Sherbrooke

Bits et vecteurs de bits — 2

Les 4 exemples suivants donnent le même vecteur x.

Exemple 1

```
signal x : bit_vector(3 downto 0);
⋮
x <= "0010";
```

Exemple 2

```
signal x : bit_vector(3 downto 0);
⋮
x <= to_bit_vector(2,4);      -- (valeur, nombre de bits)
```

Université de Sherbrooke

Bits et vecteurs de bits — 3

Exemple 3

```
signal x : bit_vector(0 to 3);
signal y : bit_vector(3 downto 0) := "0000";     -- valeur par défaut
:
:
x <= y or "0010"; -- les plages de x et y sont "compatibles"
```

Exemple 4

```
signal x : bit_vector(0 to 3);
signal y : bit_vector(3 downto 0) := (others => '0');     -- valeur par défaut
:
:
x <= y xor "0010";
```

- Attention : une valeur initiale pour un signal n'a de sens qu'en simulation.
  En synthèse, il faut procéder autrement (voir APP 2).

Université de Sherbrooke

Logique standard — 1

- Les types bit et bit_vector ne permettent que deux valeurs pour chaque position, soit '0' ou '1'.
- On s'est rendu compte éventuellement qu'il était utile en simulation et en synthèse de pouvoir représenter des états additionnels.
- Les types std_logic et std_logic_vector sont équivalents à bit et bit_vector, sauf qu'ils permettent des états additionnels en plus du '0' et du '1'.
- Ils ne font pas partie du standard de base VHDL (IEEE 1076) mais forment plutôt la base du standard dérivé de logique standard (IEEE 1164).

UDS
Université de Sherbrooke

Logique standard — 2

- La logique standard ajoute les états 'U' (uninitialized — signaux non initialisés) et 'X' (unknown — résultat d'opérations erronées).
- L'état '-' (indifférent) est utile en synthèse.
- Les autres états additionnels ('L','H','Z','W'), de même que le 'X,' servent à la résolution de conflits.
- Comme la logique standard ne fait pas partie de IEEE 1076, on y a accès en appelant explicitement la bibliothèque dans l'entête :

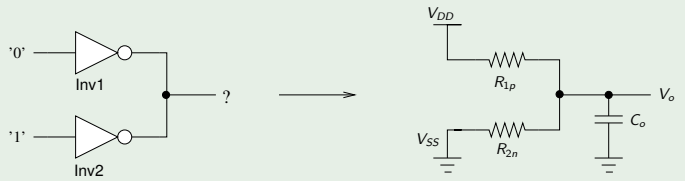
  library ieee ;
  use ieee.std_logic_1164.**all** ;
- La logique standard fait partie du IEEE 1076 à partir de VHDL 2006 (VHDL 3.0).
- Beaucoup d'outils supportent plutôt VHDL 93 ; l'entête est donc nécessaire.
- Les versions subséquentes du standard 1076 sont rétrocompatibles.

UDS Université de Sherbrooke

Conflit

Exemple : sorties raccordées et en conflit

```
y <= not x1;
y <= not x2;
```

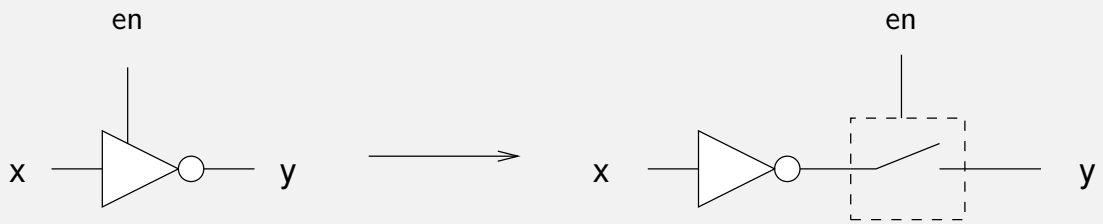


- En VHDL, le branchement de plusieurs assignations au même signal de sortie résulte normalement en une erreur.
- Toutefois, cela est permis si le type des signaux est résolu. Ceci signifie que ce type est assorti d'une *fonction de résolution de conflits*.
- C'est le cas de std_logic, mais pas de bit, ni de std_ulogic (où le 'u' veut dire unresolved).

UDS Université de Sherbrooke

Logique trois-états — 1

- Pourquoi voudrait-on modéliser une situation de conflit ?
- Logique 3-états → conflit contrôlé → bus de données
- Tampon 3-états → porte logique dont la sortie peut être "déconnectée."
- En d'autres termes, la sortie peut adopter 3 états : '0', '1' et l'état de haute impédance 'Z.'

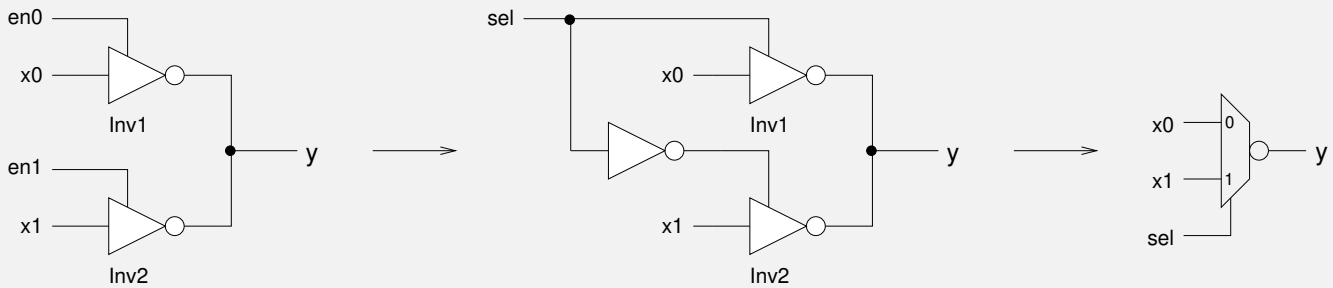


y <= **not** x when en='1' else 'Z' ;

Université de Sherbrooke

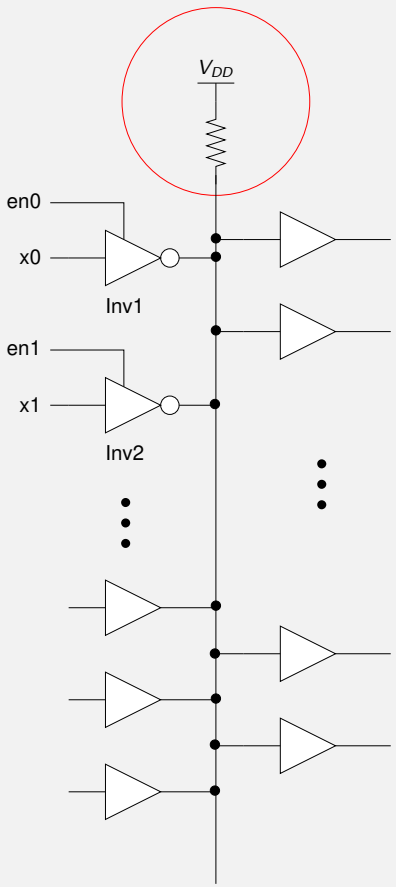
Logique trois-états — 2

- On peut brancher ensemble les sorties de plusieurs tampons pour former un bus.
- Si, à tout moment, il y a un seul tampon actif (pas en haute impédance) sur le bus, il détermine la valeur.
- On peut ainsi réaliser des multiplexeurs très simplement.




```
y <= not x0 when sel='0' else 'Z'
y <= not x1 when sel='1' else 'Z' ;
```

Bus trois-états — 1



- Un bus est un signal partagé par plusieurs circuits, en entrée comme en sortie.
- Toutes les sorties doivent être à trois états et au maximum une sortie peut être active à la fois.
- Il est souvent souhaitable de donner une valeur par défaut au bus lorsque tous les circuits l'alimentant sont en haute impédance.
- Pour ce faire, on utilise une résistance "pull-up" ('1') ou "pull-down" ('0'). La valeur de la résistance est suffisamment élevée pour ne pas nuire à une sortie active.
- Le type std_logic modélise ceci avec les niveaux logiques 'L' ('0' faible ou pull-down) et 'H' ('1' faible ou pull-up).

```
y <= not x0 when en0='0' else 'Z'
y <= not x1 when en1='1' else 'Z';
 :
y <= 'H';
```

UDS Université de Sherbrooke

Bus trois-états — 2

- Si une erreur de conception fait en sorte qu'il y a deux sorties actives sur le bus en même temps et en conflit ('0' et '1'), le simulateur donnera comme résultante le niveau logique 'X' (inconnu).
- Un conflit entre deux niveaux faibles ('L' et 'H') résulte en un niveau logique 'W' (inconnu faible).
- Les techniques à trois états sont généralement favorisées pour des bus ou multiplexeurs ayant un grand nombre d'entrées et / ou une certaine largeur en bits.
- Pour un multiplexeur, cette approche résulte en une logique plus simple, mais n'est pas forcément plus rapide (qu'un multiplexeur à base de logique).

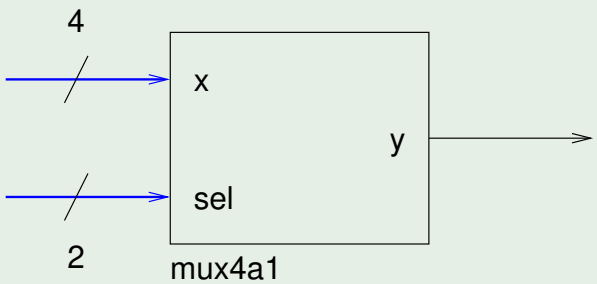
UDS
Université de Sherbrooke

Multiplexeur 4 à 1, version 3 états — 1

Exemple : multiplexeur 4 à 1, version 3 états

Entité

```
entity mux4a1 is
    port( x : in std_logic_vector(0 to 3) ;
          sel : in std_logic_vector(1 downto 0) ;
          y : out std_logic ) ;
end mux4a1 ;
```



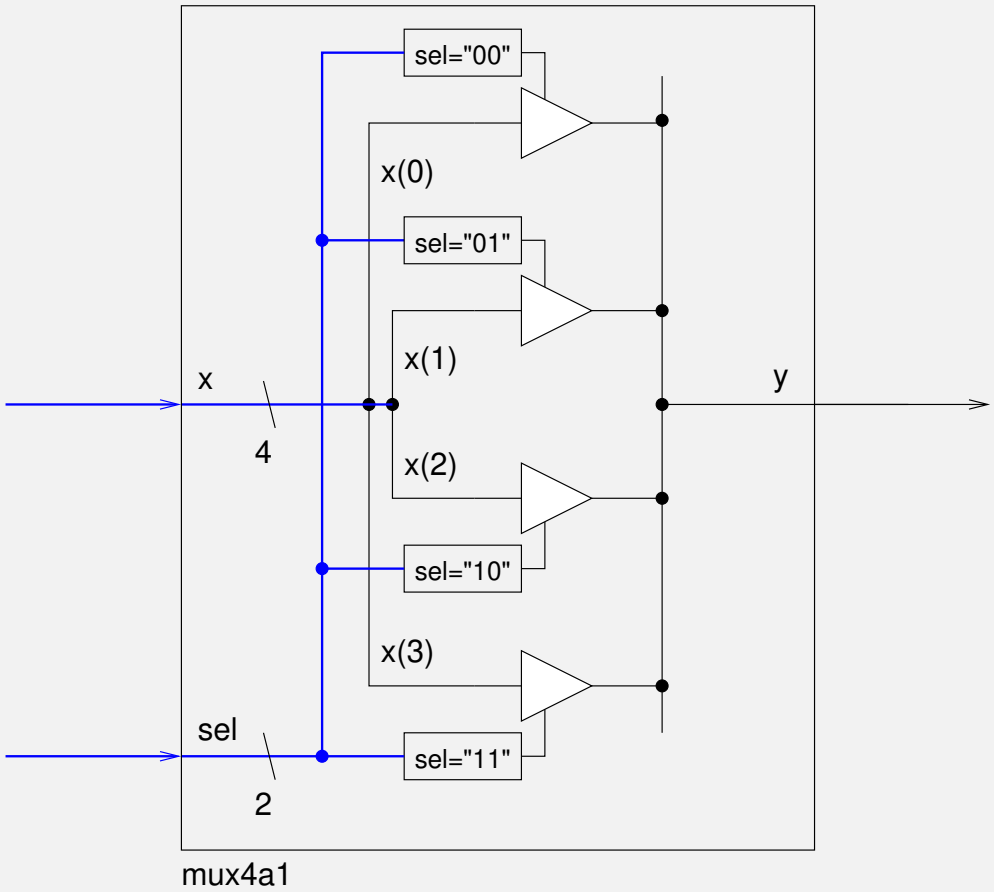
Architecture

```
architecture 3etats of mux4a1 is
begin
    y <= x(0) when sel="00" else 'Z' ;
    y <= x(1) when sel="01" else 'Z' ;
    y <= x(2) when sel="10" else 'Z' ;
    y <= x(3) when sel="11" else 'Z' ;
end architecture 3etats ;
```

Université de Sherbrooke

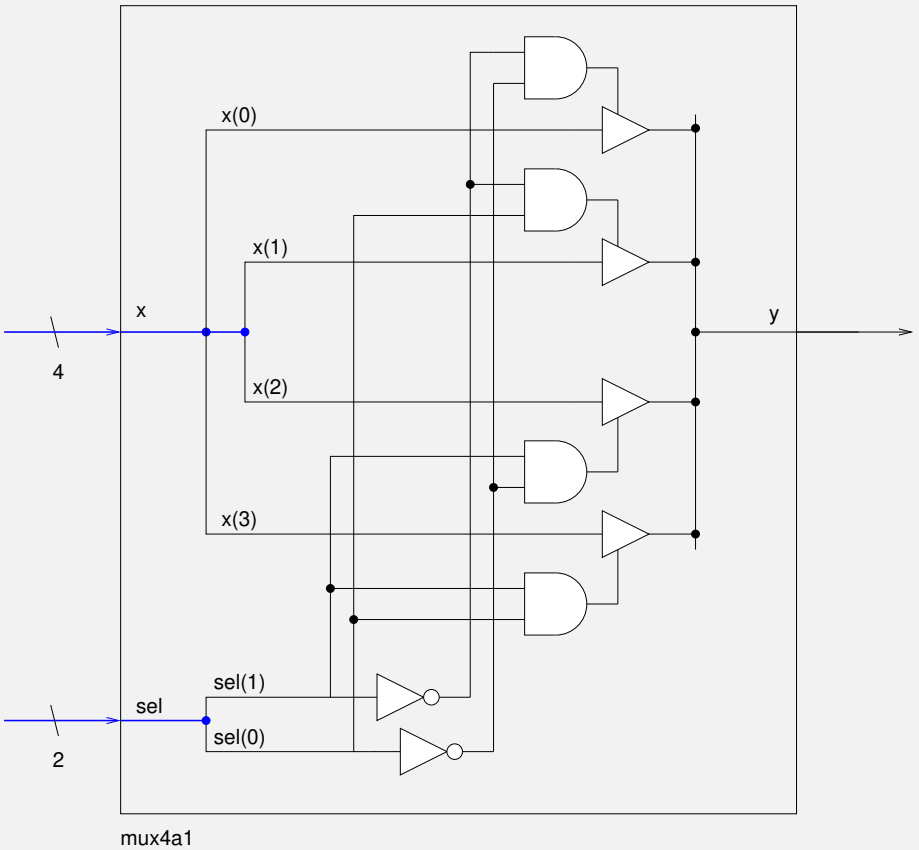
Multiplexeur 4 à 1, version 3 états — 2

Le circuit tel que décrit



Multiplexeur 4 à 1, version 3 états — 3

Ce qu'un outil de synthèse pourrait produire (il y a plusieurs possibilités)

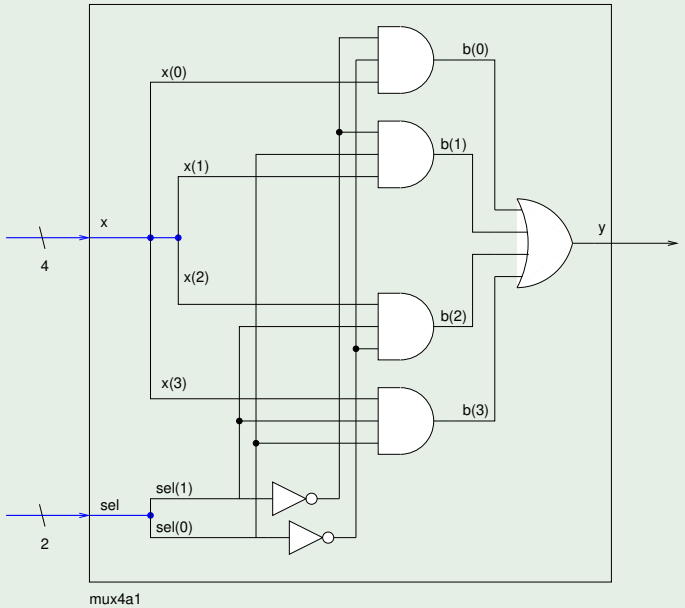


Multiplexeur 4 à 1, version portes logiques

Exemple : multiplexeur 4 à 1, version portes logiques

Architecture

```
architecture logique of mux4a1 is
    signal b : std_logic_vector(0 to 3);
begin
    b(0) <= not sel(1) and not sel(0) and x(0);
    b(1) <= not sel(1) and sel(0) and x(1);
    b(2) <= sel(1) and not sel(0) and x(2);
    b(3) <= sel(1) and sel(0) and x(3);
    y <= b(0) or b(1) or b(2) or b(3);
end architecture logique;
```



Assignation sélective

Exemple : multiplexeur 4 à 1, style comportemental

Assignation

```
architecture comport of mux4a1 is
begin
    with sel select y <=
        x(0) when "00",
        x(1) when "01",
        x(2) when "10",
        x(3) when "11",
        '0' when others;
end architecture comport;
```

Processus équivalent

```
architecture comport of mux4a1 is
begin
    muxpr : process(x, sel)
    begin
        case sel is
            when "00" => y <= x(0);
            when "01" => y <= x(1);
            when "10" => y <= x(2);
            when "11" => y <= x(3);
            when others => y <= '0';
        end case;
    end process muxpr;
end architecture comport;
```

UdS
Université de Sherbrooke

Commande **case**

Commande séquentielle

case ≪ *expression à évaluer*≫ **is**
  when ≪*valeur 1*≫ =>
    ≪*commandes séquentielles 1*≫
  when ≪*valeur 2*≫ =>
    ≪*commandes séquentielles 2*≫
      ⋮
  [**when others** =>
    ≪*commandes séquentielles n*≫]
end case;

Université de Sherbrooke

Assignation sélective

Commande parallèle

with ≪*expression à évaluer*≫ **select** ≪ *signal de sortie*≫ <=
≪*expression 1*≫ **when** ≪ *choix_1*≫,
≪*expression 2*≫ **when** ≪*choix_2*≫,
⋮
[≪*expression n*≫ **when others**] ;

Description comportementale

- La dernière version du mux 4 à 1 correspond au style de description comportemental.
- Ceci signifie que la description décrit le comportement du circuit, sans donner beaucoup de détails sur comment le réaliser.
- Les versions du mux à 3 états et à base de portes logiques sont plus spécifiques sur ce point.
- Le style comportemental permet de développer plus rapidement, à un niveau d'abstraction plus élevé.
- Cependant, ceci laisse plus de liberté à l'outil de synthèse et on a moins de contrôle sur le résultat final.

UDS
Université de Sherbrooke

Style structural

- Le style structural est l'opposé du style comportemental car il décrit la structure du circuit (un ensemble de composantes et leur interconnexions) plutôt que sa fonction.
- C'est un des grands avantages du VHDL que de pouvoir décrire la hiérarchie d'un circuit, soit sa structure en fonction d'autres descriptions de circuits.
- Ceci repose généralement sur le mécanisme des composantes.
- L'instanciation de composante est une commande parallèle qui permet d'insérer dans une description un module décrit ailleurs (correspondant à une autre description VHDL, soit une paire entité / architecture).
- Tout design VHDL moindrement complexe comprend une entité maîtresse (le "top-module") subdivisée en plusieurs modules, lesquels sont réalisés en instanciant des composantes.
- Il s'ensuit que le top-module est presque toujours de style structural.

UDS
Université de Sherbrooke

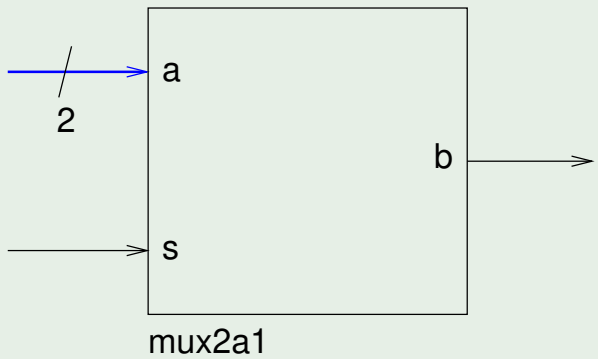
Composantes — 1

Exemple : multiplexeur 4 à 1, style structural

- On définit d'abord un multiplexeur 2 à 1 comme composante

Module VHDL

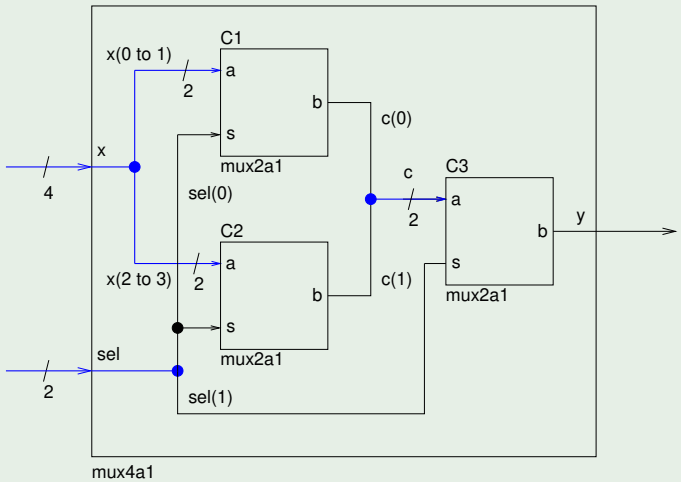
```
entity mux2a1 is
    port( a : in std_logic_vector(0 to 1);
          s : in std_logic;
          b : out std_logic );
end mux2a1;
architecture comport of mux2a1 is
begin
     with s select b <=
          a(0) when '0',
          a(1) when '1',
          '0' when others;
end architecture comport;
```



Composantes — 2

Exemple (suite) : multiplexeur 4 à 1, style structural / hiérarchique

```
architecture structure of mux4a1 is
    component mux2a1 is
        port( a : in std_logic_vector(0 to 1) ;
              s : in std_logic ;
              b : out std_logic ) ;
    end component mux2a1 ;
    signal c : std_logic_vector(0 to 1) ;
begin
    C1 : mux2a1 port map( x(0 to 1), sel(0), c(0)) ;
    C2 : mux2a1 port map( x(2 to 3), sel(0), c(1)) ;
    C3 : mux2a1 port map( c, sel(1), y) ;
end architecture structure ;
```



Déclaration de composante

Déclaration en entête de l'architecture

```
architecture structure of mux4a1 is
    component mux2a1 is
        port( a : in std_logic_vector(0 to 1);
              s : in std_logic;
              b : out std_logic );
    end component;
    signal c : std_logic_vector(0 to 1);
begin
    C1 : mux2a1 port map( x(0 to 1), sel(0), c(0));
    C2 : mux2a1 port map( x(2 to 3), sel(0), c(1));
    C3 : mux2a1 port map( c, sel(1), y);

end architecture structure;
```

Déclaration de composante

component ≪ *nom de l'entité* ≫ **is**
    port(≪ *liste des ports* ≫);
end component;

UDS
Université de Sherbrooke

Instanciation de composante

Commande parallèle

≪ *étiquette* ≫ : ≪ *nom de composante* ≫ **port map** (≪ *liste des connexions* ≫) ;

Assignation positionnelle

- L'assignation des ports peut se faire par position, c-à-d que les connexions sont énumérées dans l'ordre où les ports sont déclarés.

C1 : mux2a1 **port map**(x(0 **to** 1), sel(0), c(0)) ;

Assignation nommée

- L'assignation peut aussi se faire en nommant explicitement les ports, auquel cas l'ordre d'assignation n'a plus d'importance.

C1 : mux2a1 **port map**(s => sel(0), a => x(0 **to** 1), b => c(0)) ;

UDS Université de Sherbrooke

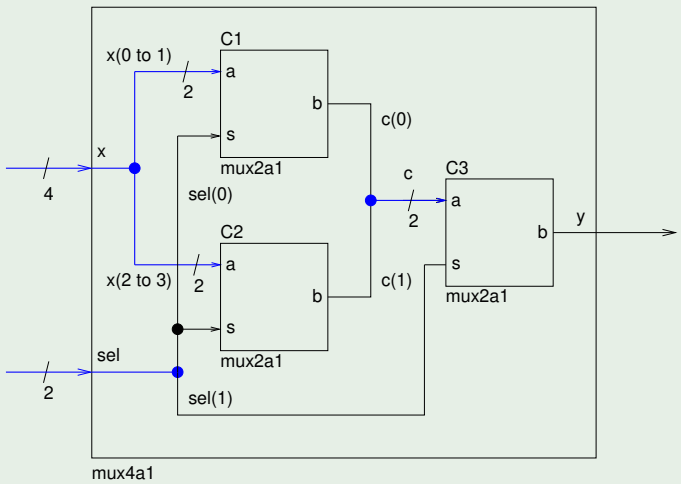
Instanciation de composante — instanciation directe

Définition

L'instanciation directe ou l'instanciation d'entité est une méthode alternative d'instanciation qui permet d'omettre la déclaration de composante (disponible seulement depuis VHDL 93).

Exemple : mux 4 à 1, style structural, instanciation directe

```
architecture structure of mux4a1 is
  signal c : std_logic_vector(0 to 1);
begin
  C1 : entity work.mux2a1(comport)
   port map(a=>x(0 to 1), s=>sel(0), b=>c(0));
  C2 : entity work.mux2a1(comport)
   port map( x(2 to 3), sel(0), c(1));
  C3 : entity work.mux2a1(comport)
   port map( c, sel(1), y);
end architecture structure;
```



Principe d'"incertitude" en VHDL — 1

Principe d'"incertitude"

Pour toute condition non spécifiée, le comportement par défaut assumé en VHDL est de maintenir la valeur de sortie.

- Il s'ensuit qu'une condition non-spécifiée implique la génération d'un élément de mémoire (bascule).
- La bonne pratique veut donc que le comportement soit spécifié pour tous les cas.

UDS Université de Sherbrooke

Omission d'un signal dans la liste de sensibilité

Exemple

```
somme : process(a, b)
begin
    s <= a xor b xor re;
end process somme;
```

- Le signal re ne fait pas partie de la liste de sensibilité, donc VHDL voudra maintenir la valeur de s s'il y a un changement sur re.
- Il ne s'agit donc plus de logique purement combinatoire.

UDS Université de Sherbrooke

Omission de choix — commande **if**

Exemple

```
if S='0' then
    Y <= A0;
end if;
```

- Pour S ayant une valeur autre que '0', la valeur de Y est maintenue.

Exemple ambigu

```
if S='0' then
    Y <= A0;
elsif S='1' then
    Y <= A1;
end if;
```

- On semble couvrir tous les choix.
- Si S est de type std_logic, il peut prendre 9 valeurs.
- Le comportement du simulateur ou de l'outil de synthèse est incertain.

- Bonne pratique : inclure une clause **else** englobant tous les cas restants.
- Le même constat s'applique à l'assignation conditionnelle.
- Pour les commandes **case** et les assignations sélectives, inclure une clause **when others**.

UDS
Université de Sherbrooke

Section 4

Récapitulatif

Récapitulatif — 1

- VHDL est un langage de modélisation de systèmes numériques et non un langage de programmation.
- Il permet de travailler à différents niveaux d'abstraction, mais se situe à plus bas niveau généralement qu'un langage de programmation conventionnel.
- Le corps d'une architecture est un contexte parallèle, tandis que le corps d'un processus est un contexte séquentiel.
- Il y a deux sous-ensembles distincts de commandes : les commandes parallèles et les commandes séquentielles.
- VHDL permet de décrire le comportement ou la structure d'un circuit (ou un mélange des deux).

UDS
Université de Sherbrooke

Récapitulatif — 2

- On n'utilise pratiquement plus les schémas aujourd'hui, sauf pour des circuits très simples.
- L'utilisation d'un des 2 grands langages de description de matériel (VHDL ou Verilog) est donc incontournable.
- VHDL permet de modéliser certains effets analogiques non-idéaux.
- Ceci permet entre autres de modéliser de la logique à 3 états (avec std_logic).
- Plusieurs outils supportent la version VHDL-93 du standard.
- Attention au principe d'incertitude.

UDS
Université de Sherbrooke