Unités : CSN & SysLog2

Le langage VHDL, notions de base & synthèse

HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Etienne Messerli

Septembre 2021

Documents

- Manuel VHDL, synthèse et simulation, E.Messerli, HEIG-VD
- Livres recommandés:
  - VHDL. Introduction à la synthèse logique. Philippe Larcher, Eyrolles, 1997
  - Le langage VHDL. Du langage au circuit, du circuit au langage. J. Weber et M. Meaudre, Dunod, 2001
  - Le langage VHDL. Du langage au circuit, du circuit au langage. 5ème édition. J. Weber/ S. Moutault/M. Meaudre, Dunod, 2016

- Reference guide:
  - The VHDL Golden Reference Guide, compatible IEEE std 1076-2002
    disponible chez : Doulos, http://www.doulos.com/

Documents

- RTL Hardware design using VHDL, PONG P. CHU Wiley, 2006.
  - Disponible sur Safari books online :
    https://proquest.tech.safaribooksonline.de/9780471720928

Manuel VHDL, synthèse et simulation

Liste des chapitres du "Manuel VHDL" :

- 1. Introduction
- 2. Les concepts du langage VHDL
- 3. Les instructions concurrentes

  paragraphes 3-1 à 3-4
- 7. Visseries et astuces

  paragraphes 7-1 à 7-4
- 9. Description de systèmes combinatoires

Evolution VHDL 2008 (IEEE 1076-2008)

- Cours : utilisation norme **VHDL-2008** NEW 
  - utilisation partielle dans le cadre du cours CSN/ SysLog2
- VHDL 2008:
  - Version 2008 incompatible version 1993
  - Synthèse: support suffisant du VHDL 2008
  - Simulation : support complet avec QuestaSim
- Evolutions VHDL 2008 pour la synthèse:
  - opérateur logique: std_logic operateur vecteur
  - opérateur logique unaire: opération logique bit à bit du vecteur
  - affectation hexadécimal: possible pour taille autre que multiple de 4
  - **déclaration "all"** pour liste sensibilité d'un process
    - description d'un système combinatoire
  - …

Version VHDL utilisée au cours

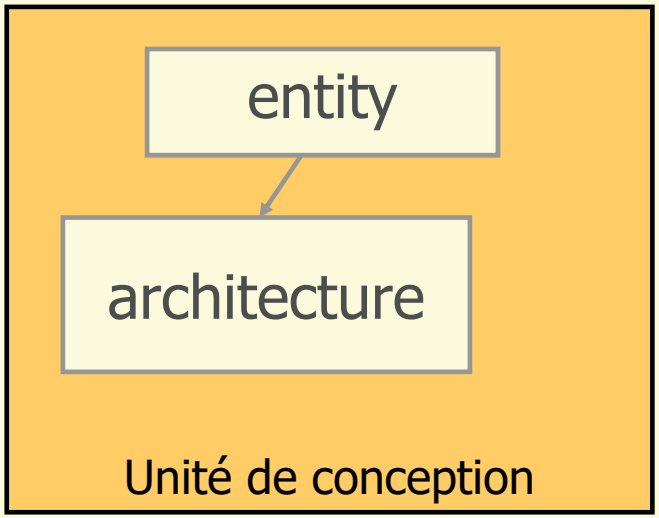
- Utilisation norme **VHDL-2008**

  - utilisation très partielle dans le cadre des cours CSN/ SysLog2
  - seul la déclaration **"all"** pour l'instruction process est supporté par le synthétiseur Quartus
  - autrement : utilisation de la syntaxe VHDL-93
    - compatible avec VHDL-2008

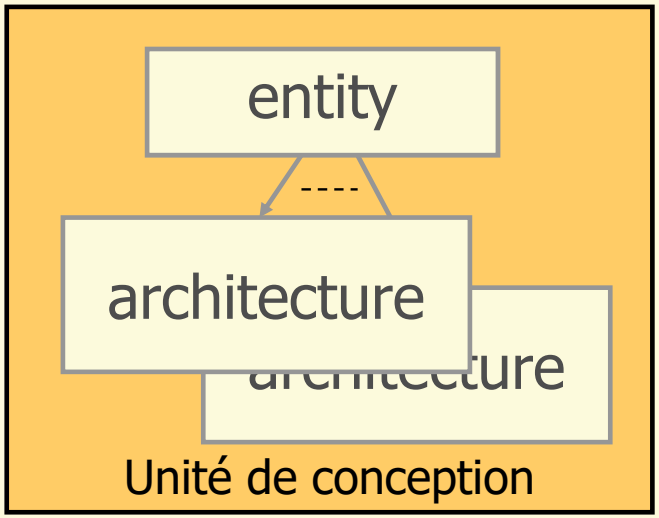
Unité de conception (module VHDL)

- Une entité (*entity*)
- Une ou plusieurs architectures (*architecture*)
- Une configuration
- Des bibliothèques et des paquetages

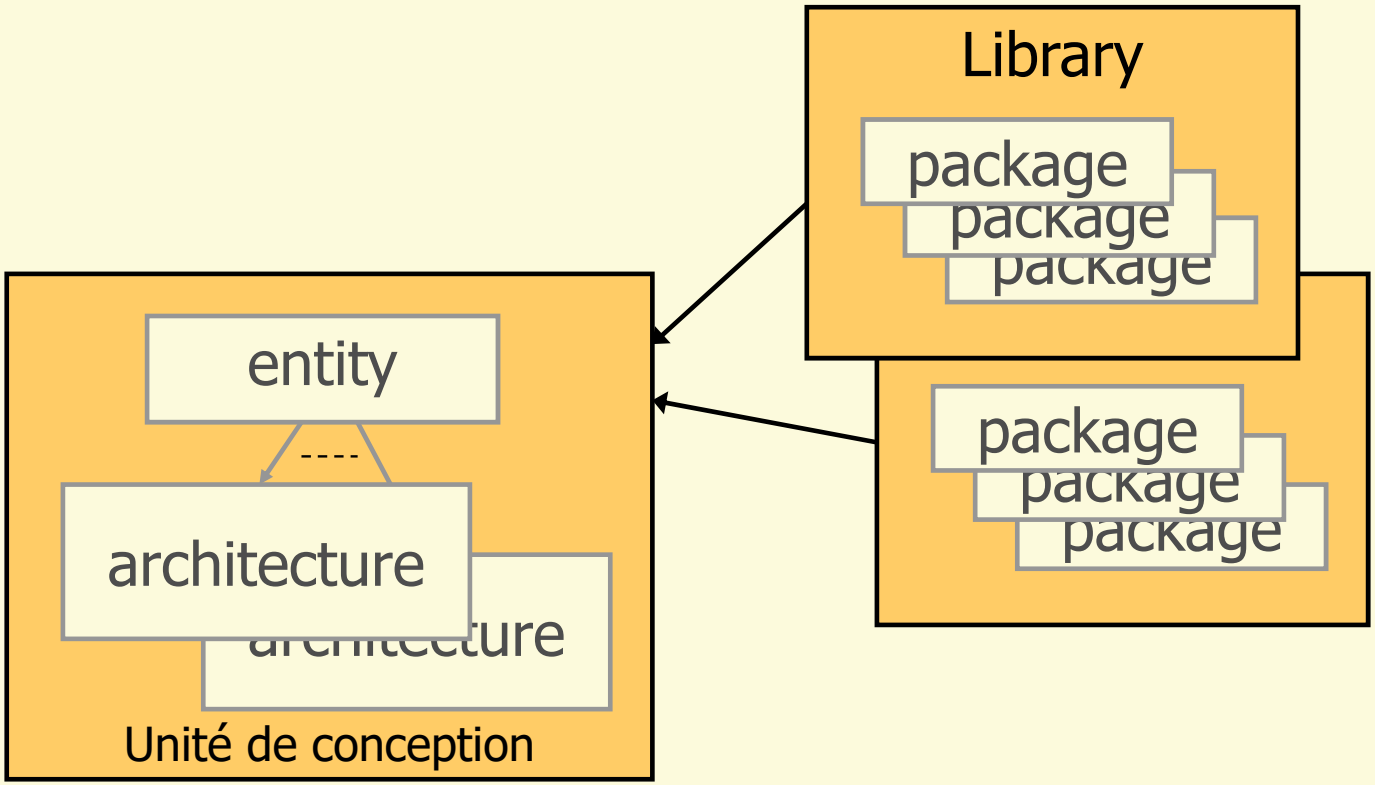
… unité de conception …



… unité de conception …

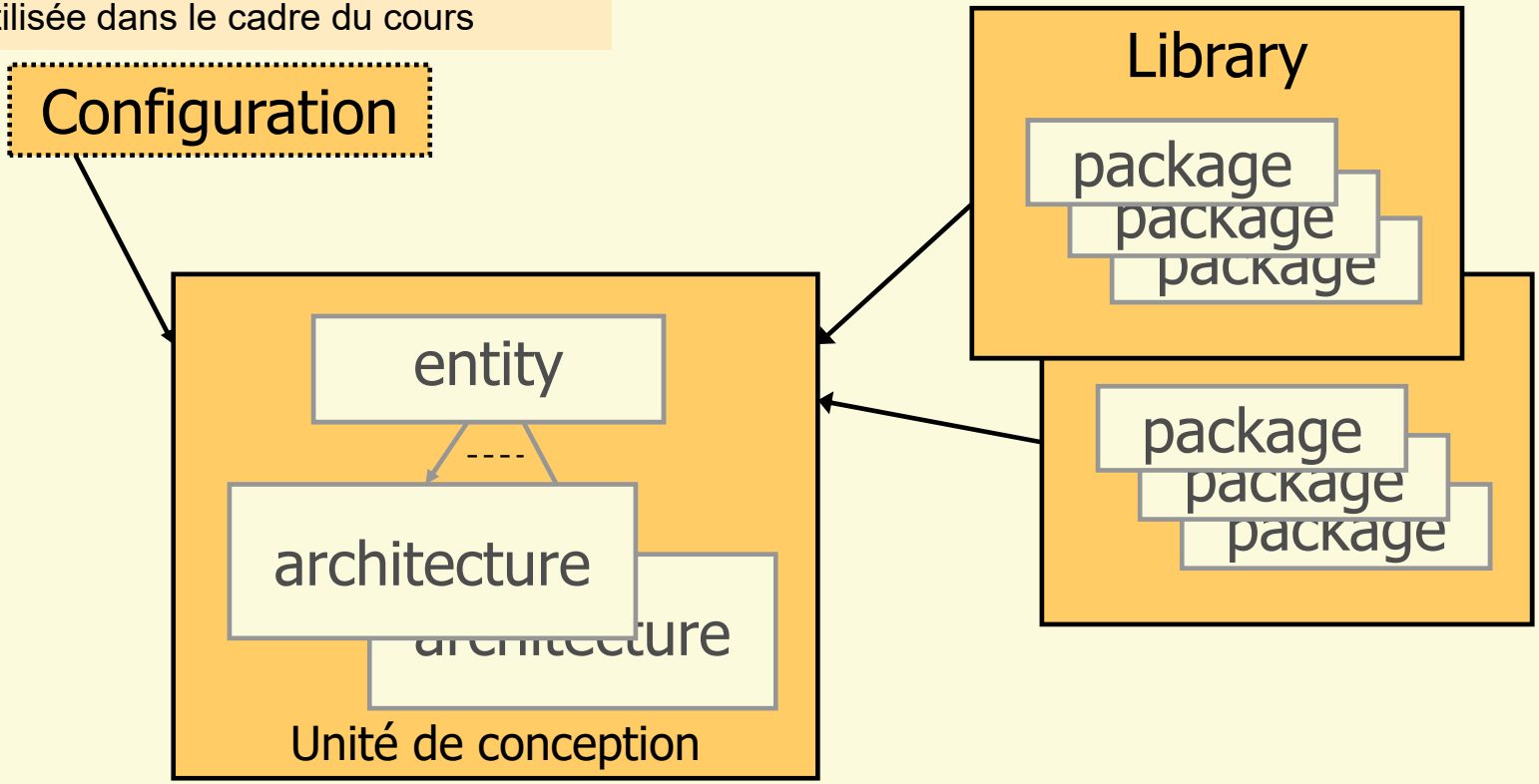


… unité de conception …



… unité de conception …

Configuration de multiple architectures
- pas utilisée dans le cadre du cours



Les éléments du langage …

- Mots clés : mots réservés du langage

  le langage définit 100 mots réservés (voir annexe)

- Instructions VHDL terminées par le séparateur « ; »

```
-- exemple instruction terminée par ;
sortie_o  <= (A_i or not B_i) and C_i;
```

… les éléments du langage …

- Commentaires :
  débutent par deux tirets « -- » et se terminent avec la ligne

```
--Ceci est un commentaire
--sur deux lignes

entity exemple is              -- commentaire ..
  port (A : in std_logic;      -- commentaire ..
        ...
```

… les éléments du langage …

- Identificateurs :
  - ils doivent commencer par une lettre
  - ils comportent des lettres (minuscules ou majuscu-les), des chiffres et des soulignés (*underscore*)
  - un seul souligné (*underscore*) de suite : `bus_data_interne`
  - le VHDL ne distingue pas les majuscules des minuscules (***not case sensitive***)
  - exemple d'identificateurs identiques :

    `bus_data, Bus_Data, BUS_DATA, BuS_DatA`

Les objets du langage

Objet : information manipulée par le langage

Ils sont répartis en quatre classes :

- **constantes** : objets de valeurs fixes
- **variables** : objets appartenant au monde ***software***
- **signaux** : objets appartenant au monde ***hardware***
- **fichiers** : objets servant à stocker des valeurs

On parle fréquemment de **classes** d'objets

Les types du langage

- scalaires :
  - énumérés (boolean, std_logic,..),
  - entiers (integer, ...), physiques (time),
  - flottant (real)
- composites : tableau (array)

  type std_logic_vector **is** array(natural range<>) **of** std_logic;

- *accès : pointeur pour accéder à des objets (dynamique)*
- *fichier : séquence de valeur d'un type donné*

Affectation d'un signal

- Syntaxe de l'affectation d'un signal :

```
signal  <=  expression;
```

L'affectation représente un lien **définitif** entre le signal et le circuit générant l'expression (connexion)

L'affectation du signal ne modifie pas la valeur courante mais les valeurs **futures**

… affectation d'un signal

- Affecter une expression à un signal correspond à connecter un signal sur la sortie d'une porte.
- Plusieurs affectations sur un même signal correspond à un

court-circuit !

Affectation de variable

Pas utilisé dans ce cours. Variables seront utilisées dans les cours à choix

- Syntaxe de l'affectation d'une variable :

```
variable  :=  expression;
```

L'affectation de la variable est instantanée, ensuite il n'existe plus aucun lien entre la variable et l'expression

Le type std_ulogic (type énuméré)

Défini par le paquetage ieee.std_logic_1164

```
type std_ulogic is (
      'U',-- état non initialisé
      'X',-- état inconnu fort
      '0',-- état logique 0 fort
      '1',-- état logique 1 fort
      'Z',-- état haute impédance
      'W',-- état inconnu faible
      'L',-- état logique 0 faible
      'H',-- état logique 1 faible
      '-' -- état indifférent, don't care   );
```

std_logic et std_ulogic

- **std_ulogic** : L'interconnexion entre deux signaux est interdite (unresolved).
  - cas des sorties standards des circuits numériques. Elles ne peuvent pas être connectées ensembles.
- **std_logic** : Interconnexion possible grâce à l'utilisation d'une fonction de résolution
  - correspond aux sorties spéciales (trois états, collecteur ouvert, ..) qui sont interconnectées.

Type std_logic généralement utilisé dans l'industrie
=> utilisé pour le cours

Fonction de résolution pour type std_logic

- La valeur affectée au point d'interconnexion de deux sorties std_logic est régie par la table de résolution suivante:

```
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' )  -- | - |
                                                    );
```

Fonction de résolution pour type std_logic

- La valeur affectée au point d'interconnexion de deux sorties std_logic est régie par la table de résolution suivante:

```
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' )  -- | - |
                                                    );
```

Fonction de résolution pour type std_logic

- La valeur affectée au point d'interconnexion de deux sorties std_logic est régie par la table de résolution suivante:

```
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' )  -- | - |
                                                    );
```

Type utilisé dans l'industrie :

Le type le plus couramment utilisé dans l'industrie est le

std_logic

Attention: en cas d'interconnexion erronée alors

- pas de verrouillage par le langage VHDL
- détecté en simulation par un état **'X'**
- détecté lors de la synthèse par une erreur : multiples *drivers*

Déclaration d'un signal std_logic

- Syntaxe de la déclaration

```
signal A : std_logic;
```

A l'instant t=0ns le signal aura l'état non initialisé 'U' (*Uninitialized*)

- Possible de donner une valeur initiale

```
signal A : std_logic := '0';
```

Cette initialisation est purement "soft", elle est exécutée à l'instant t=0 ns.

Pas utilisable en SYNTHESE

Le type std_logic_vector

- Le type `sdt_logic_vector` est défini dans le paquetage
  `ieee.std_logic_1164`
- Celui-ci est basé sur le type `sdt_logic` :

```
type std_logic_vector is
    array(natural range<>) of std_logic;
```

- Déclaration d'un signal :

```
signal vecteur : std_logic_vector(7 downto 0);
```

Déroulement concurrent et séquentiel

- Dans un langage informatique :
  - les instructions ont un déroulement séquentiel
- Dans un circuit :
  - toutes les portes fonctionnent simultanément
  - tous les signaux évoluent de manière concurrente
- Le langage VHDL dispose d'instructions concur-rentes pour la description de circuits (matériel)

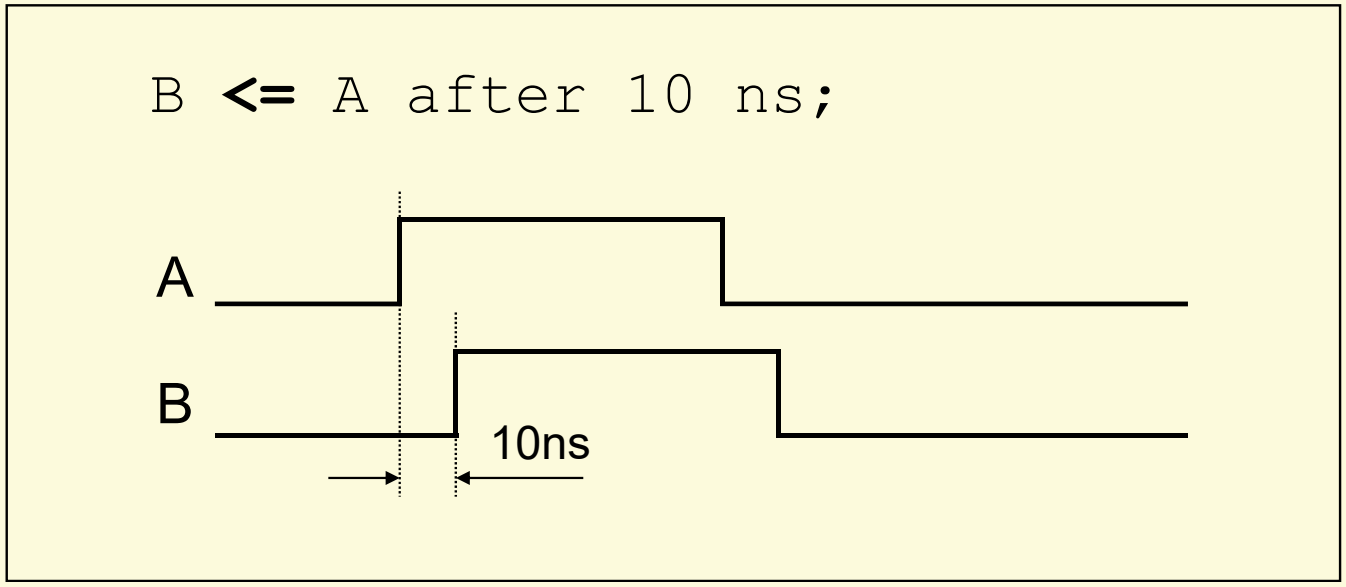
La notion de temps ...

- Le VHDL définit la notion de temps
  - indispensable pour la description de matériel
  - inconnue des précédents langages tels : CUPL, ABEL, ..
- Possible de modéliser le comportement réel des portes logiques (temps de retard)
- Synthèse, simulation, modélisation et spécification :

même langage !

… notion de temps …

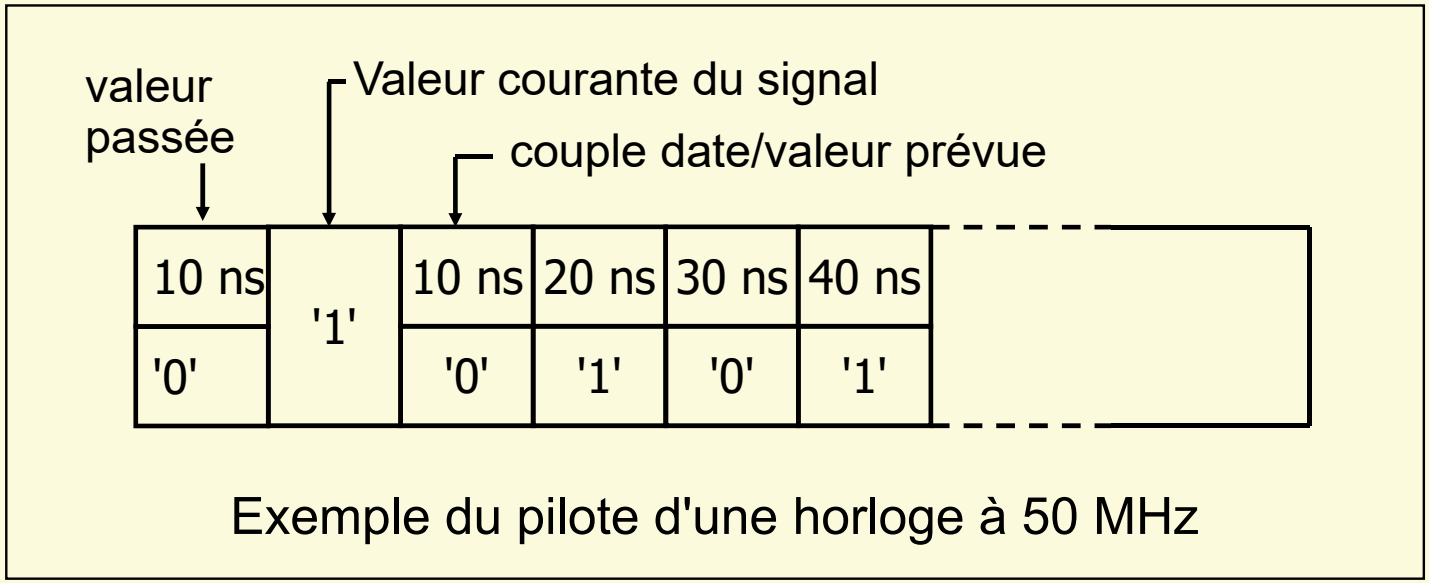
- Permet de simuler l'écoulement du temps
- Intégré dans la notion de signal :



Le pilote d'un signal

La notion de temps permet d'exprimer l'évolution d'un signal

=> nous parlons du PILOTE d'un signal



Exemple du pilote d'une horloge à 50 MHz

Les opérateurs du langage

- Il y a sept classes d'opérateurs :
  - opérateurs logiques
  - opérateurs relationnels
  - opérateurs de décalage (VHDL93)
  - opérateurs d'addition
  - opérateurs de signe
  - opérateurs de multiplication
  - opérateurs divers

Les 7 classes d'opérateurs

- Logiques : **`and, or, nand, nor, xor, xnor`**
- Relationnel : **`=, /=, <, <=, >, >=`**
- Décalage : **`sll, srl, sla, sra, rol, ror`** (VHDL93)
- Addition : **`+, -, &`** (concaténation)
- Signe : **`+, -`**
- Mult./Div. : **`*, /, mod, rem`**
- Divers : **`**, abs, not`**

Il est possible de surcharger les opérateurs

Fonctionnalité utilisée dans les librairies standards ieee

Types pour les opérateurs

- Logiques : Bit ou Boolean
- Relationnel : tous les types, résultat Boolean
- Décalage : Bit ou Boolean, décalage Integer
- Addition : type numérique (Integer, Natural,..)
- Concaténation : type Array
- Signe : type numérique (Integer, Natural,..)
- Mult./ Div. : type numérique (Integer, Natural,..)
- Divers : type numérique (Integer, Natural,..)
  - not : idem opérateurs logiques

Ces opérateurs sont définit dans le paquetage standard

VHDL, un langage typé …

- Exemple d'expression erronée :

```
   signal A, B, Y : std_logic;
begin
    Y <= (A >= B);
end;
```

Résultat expression `(A >= B)` : booléen (`true` ou `false`)

Pas du même type que signal Y : type `std_logic`.

… VHDL, un langage typé

- Exemple d'expression correcte :

```
   signal A, B, Y : std_logic;
begin
    Y <= '1' when (A >= B) else '0';
end;
```

La valeur affectée au signal `Y` est bien du type `std_logic`.

L'instruction `when … else` demande une condition de type booléen (`true` ou `false`).

… VHDL, un langage typé

- Exemple d'expression adaptée en VHDL-2008 :

```
signal A, B, Y : std_logic;
begin
    -- VHDL 2008
    Y <= (A ?>= B);
end;
```

Actuellement pas supporté par Quartus 18.1 !!!

Affectation std_logic / std_logic_vector

- Affectation std_logic correspond à assigner un caractère :

```
signal_1bit <= '1';
```

simples guillemets

- Affectation std_logic_vector correspond à assigner une chaîne de caractères :

```
vecteur_4bits <= "0101";
```

doubles guillemets

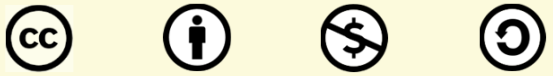
vide volontairement

Le VHDL pour la synthèse automatique

HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Etienne Messerli

REDS@HEIG-VD



Fausses idées sur le langage VHDL

Le langage en tant que tel ne garantit pas :

- qualité des descriptions
- portabilité des descriptions
- descriptions synthétisables
- design optimum (quantité de logique)
- performance élevée (fréquence maximum)

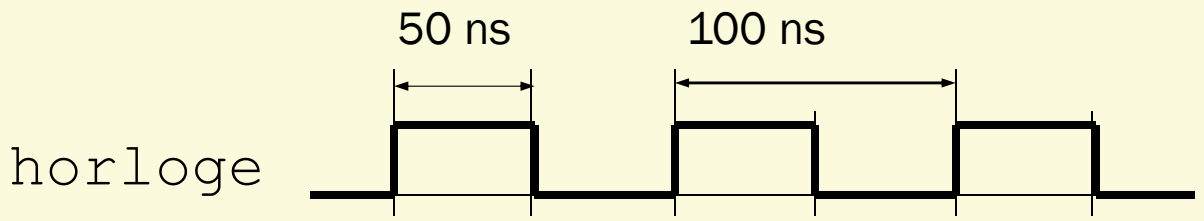
Mauvais concepteur + VHDL = **catastrophe**

Exemple de description VHDL

Soit la description VHDL suivante :

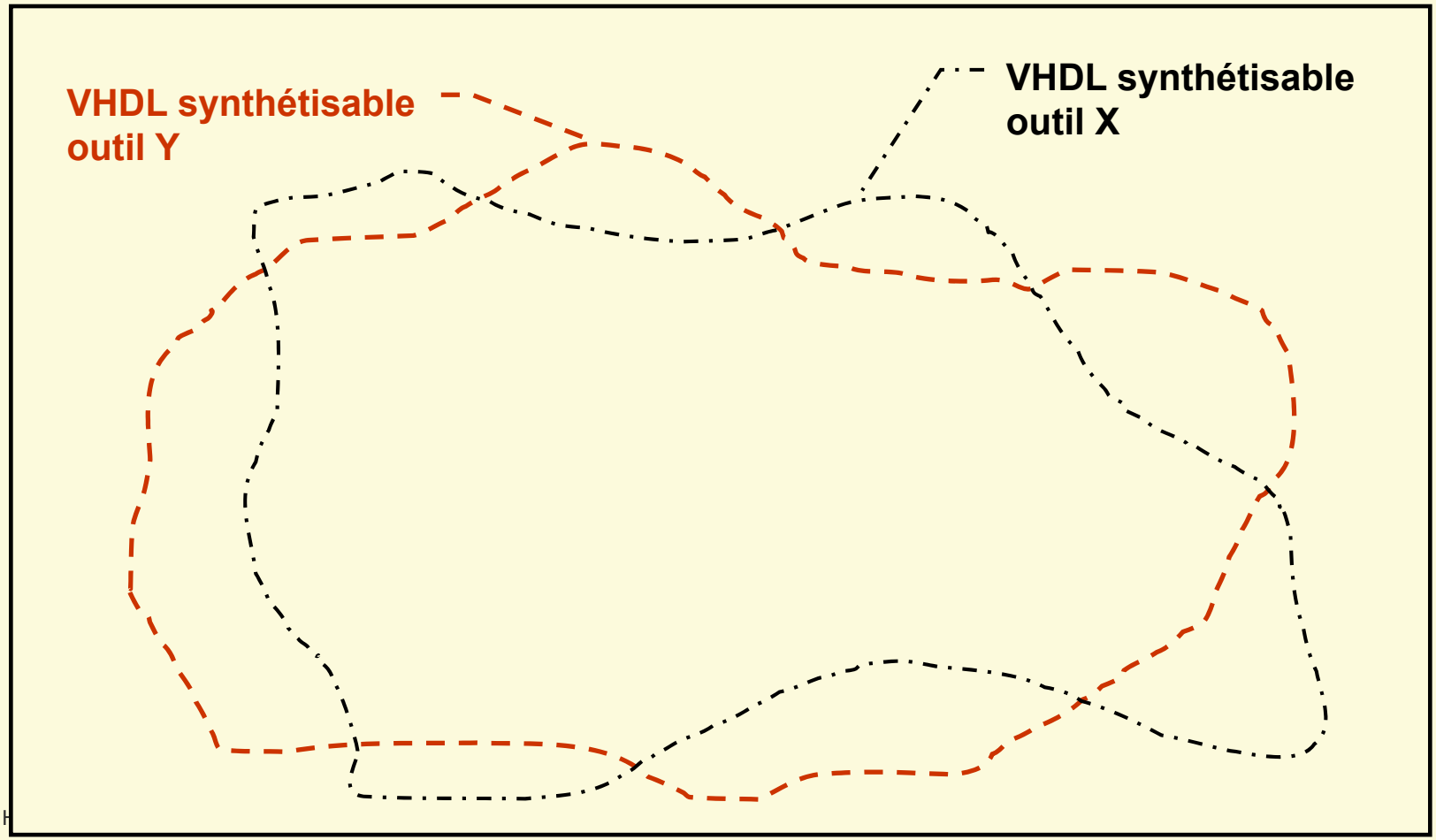
```
horloge <= not horloge after 50 ns;
```

Cela correspond au chronogramme suivant :

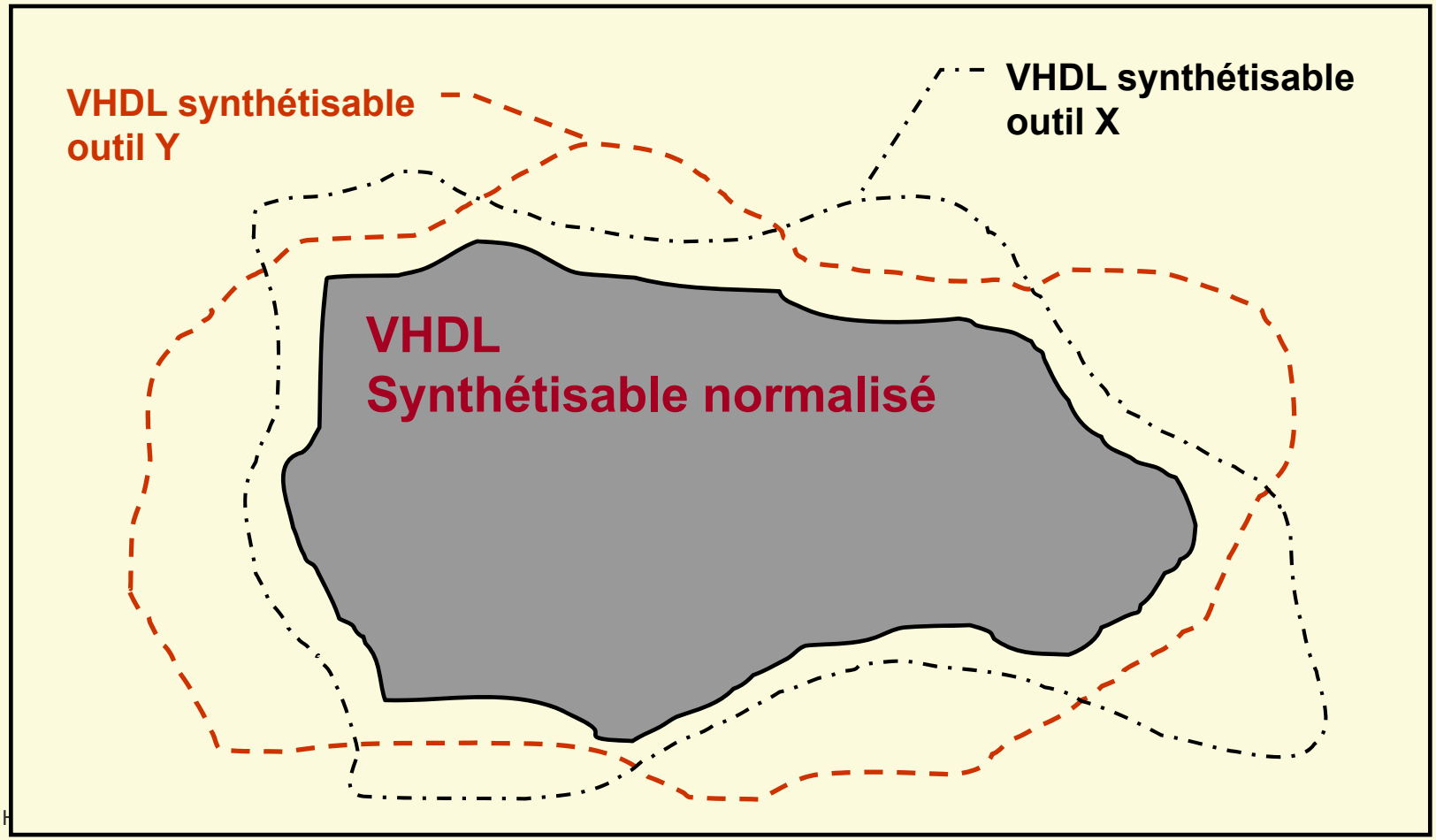


Il s 'agit d 'une horloge à 10Mhz

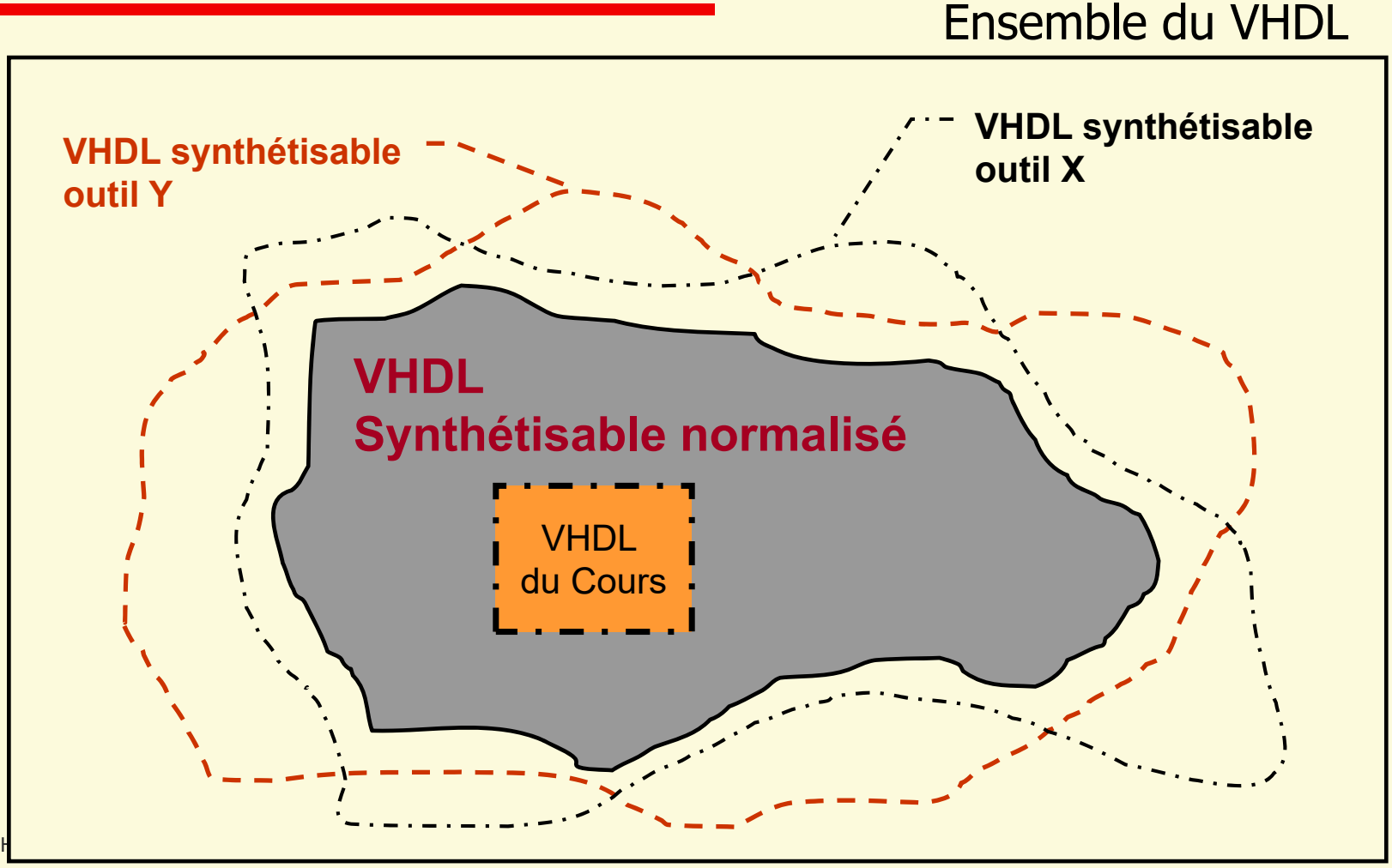
Ensemble synthétisable du VHDL



Ensemble synthétisable du VHDL



Ensemble synthétisable du VHDL



Conception avec le VHDL

- Il faut penser CIRCUIT.
- Une bonne conception commence par une décomposition du système (hiérarchie).
- Il faut imaginer l'architecture physique du système
  => pour concevoir design efficient.
- Le VHDL **n'est pas** un outil de CONCEPTION.
- Le VHDL est un langage de **DESCRIPTION**.

HDL = Hardware Description Language

Apprentissage du langage

- Connaissance des instructions VHDL seules est insuffisante
- Ensemble du langage VHDL pas nécessaire
- Syntaxe des instructions pour la synthèse
- Différence entre simulation et synthèse
- Méthodologie

Formation sur le langage INDISPENSABLE

Structure d'un module VHDL

- Un module VHDL est composé de :
  - référence à des bibliothèques :
    - `ieee.std_logic_1164` définit type std_logic et opérateurs de base
    - `ieee.numeric_std` opérations arithmétiques
    - utilisation d'une bibliothèque propre, maitrise de la source (recommandé)
  - entité (entity) :
    - définit les connexions du module avec l'extérieur
  - architecture (architecture) :
    - définit le fonctionnement interne
    - possible d'avoir plusieurs architectures

Convention REDS pour les identificateurs

Similaire programmation en C

- Déclaration dans une entité :

```
<nom_signal>_i    pour une entrée (in)
<nom_signal>_o    pour une sortie (out)
```

- Déclaration dans une entité uniquement au top :

```
n<Nom_signal>_i   pour signal actif bas (polarité négative)
n<Nom_signal>_o   pour signal actif bas (polarité négative)
<nom_signal>_io   pour une entrée/sortie (inout)
```

- Déclaration dans une architecture :

```
<nom_signal>_s    pour un signal interne
<NOM_CONST>       pour une constante tout en majuscule
```

- *Déclaration dans un processus :*

```
<nom_variable>_v  pour une variable
```

Structure de l'entité (entity)

```
-- Librairie ieee
library ieee;
  use ieee.std_logic_1164.all; --Defini type std_logic

entity exemple is
  port(entree_i   : in  std_logic;
       vecteur_i : in  std_logic_vector(3 downto 0);
       sortie_o   : out std_logic;
       bidir_io   : inout std_logic);
end exemple;
```

Structure de l'architecture

```
architecture style_description of exemple is
  --zone de déclaration
begin
  --Instructions concurrentes  .......
  process (liste_de_sensibilité)
  begin
     --Instructions séquentielles  .......
  end process;
end style_description;
```

Styles de description RTL

Styles de descriptions	Abréviations
• Equations logiques	logique
• Table de vérité	tdv
• Flot de données	flot_don
• Comportementale	comport
• Machine d'états	m_etat
• Structurelle (schéma bloc)	struct
• ….	

Architecture: zone de déclaration

- Déclaration de signaux internes

```
signal interne_s : std_logic;
signal vect_s    : std_logic_vector(4 downto 0);
```

- Déclaration de constantes

```
constant LIMITE : std_logic_vector(2 downto 0):="101";
```

- Déclaration de composants
- Déclarations de types, de procédures et de fonctions
  (pour utilisateurs expérimentés)

Architecture, zone de description

- Déroulement concurrent :
  - toutes les instructions concurrentes
  - les processus (instr. concurrente !)
- Si plusieurs processus :
  - exécution concurrente entre les processus

- Déroulement séquentiel :
  - **UNIQUEMENT** à l'intérieur d'un processus

Les instructions concurrentes

- Affectation simple

  Y <= … oper_logic …;

- Affectation conditionnelle

  Y <=.. when .. else .. ;

- Instruction de sélection

  with .... select Y <= …. when ….

- Instanciation de composants (port map).
- Processus (process)

Instruction d'affectation …

- Syntaxe de l'affectation simple :

```
signal <= expression;
```

Exemples d'expression :

```
expression Operator expression
identifier
aggregat
…
```

Exemples d'affectation simple:

```
sortie  <= sel0 or sel1 or sel2;
signal  <= vect_8bits(3);
vect(0) <= (A or B) and C;
signal  <= entree;
```

… instruction d'affectation …

- Syntaxe de l'affectation conditionnelle
  - cas avec une seule condition

```
signal1 <= expression_true when condition else
           expression_false;
```

Exemples :

```
egal <= '1' when (valeur="1001") else
        '0';

result <= A or B when (c='1') and (en='1') else
          '0';
```

… instruction d'affectation

- Syntaxe de l'affectation conditionnelle avec plusieurs conditions :

```
signal1 <= expression when cond_booleen else
           expression when cond_booleen else
           ...
           expression;
```

Dessiner le schéma logique équivalent de cet exemple :

```
result <= '1'      when (nForce1 = '0') else
          A and B when (fct_and = '1') else
          A ;
```

Remarque : L'instruction when…else a une notion **de priorité** entre les différentes conditions

Instruction de sélection …

- Syntaxe de l'instruction de sélection :

```
with signal_commande select
  signal_affecte <= expression1 when "com_etat1",
                    expression2 when "com_etat2",
                                ...
                    expressionN when others;
```

Remarque : le terme **others** définit **toutes les autres** combinaisons possibles de l'état du signal de commande (! type std_logic !)

… instruction de sélection

- Exemple :

```
with val_sel select
  sortie <= entr_A when "00",
            entr_B when "01",
            entr_C when "10",
            entr_D when "11",
            'X'    when others; --simulation
```

Remarque : le terme others représente les combinaisons "UH", "ZZ", "X0", … du signal `val_sel`

Instanciation d'un composant …

- Dans la zone déclaration de l'architecture :

```
Component nom_composant is -- is accepte en VHDL93
  port (ports_entree : in std_logic;
            .....
        ports_sortie : out std_logic);
end component;
for all : nom_composant use
          entity work.nom_entity(style_description);
```

- Dans la zone de description de l'architecture :

```
[Label:] nom_composant port map
          (signal_entrees => signal_achitecture1_s,
           signal_sorties => signal_achitecture2_s);
```

... instanciation d'un composant ...

```
architecture struct of exemple is
  component porte_et is -- is accepte en VHDL93
            port (A_i, B_i : in  std_logic;
                  Z_o      : out std_logic);
  end component;
  for all : porte_et use
              entity work.porte_et(flot_don);
begin
   ..
  U1: porte_et port map (A_i => entr_i,
                         B_i => signal_s,
                         Z_o => sortie_o);
   ..
end struct;
```

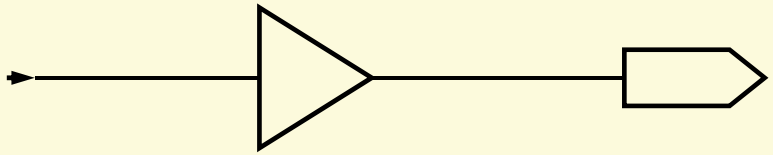
… instanciation d'un composant

- Cas d'une sortie non utilisée (unconnected) :

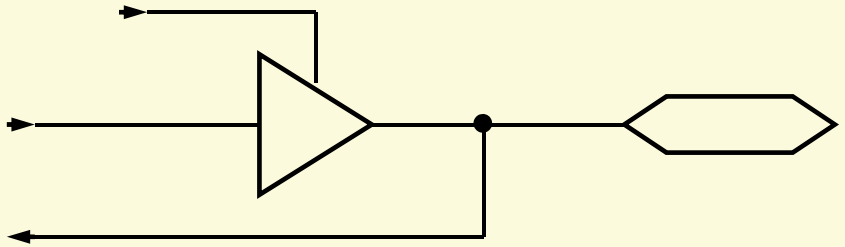
```
architecture struct of exemple is
  component module
            port (  …
                  sortie_o   : out std_logic
                    …        );
  end component;
  for all : module use entity  work.module(comport);
begin
   ..
  U1: module port map (   …
                        sortie_o => open,
                        --sortie_o => pas connecté
                           …              );
   ..
end struct;
```

Types de *port* ...

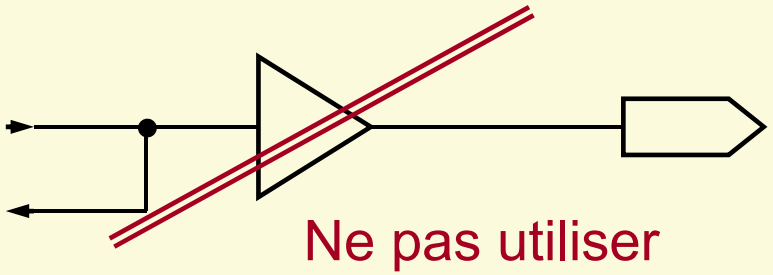
`port` **`out`**



`port` **`inout`**

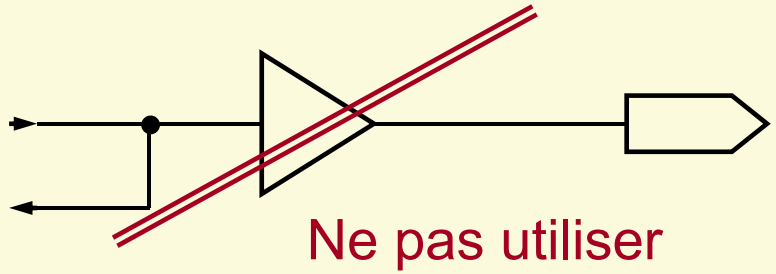


`port` **`buffer`**

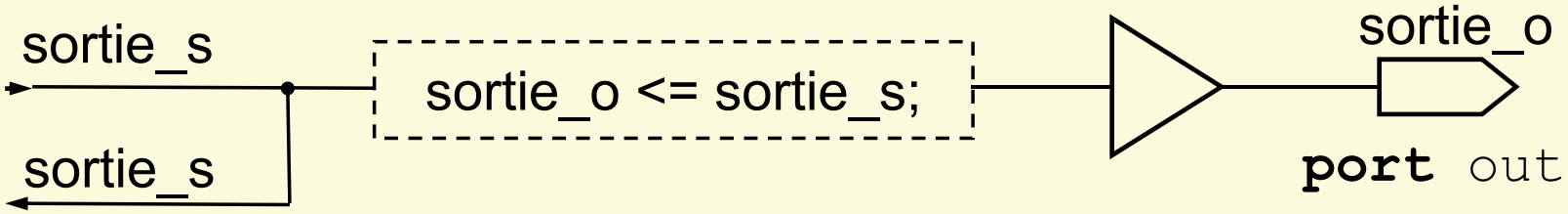


… types de port

`port` **`buffer`**



Remplacer par un signal interne et le type de port **out**, soit:



Remplacement du type: **buffer**

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

entity exemple is
  port( A_i, B_i : in  std_logic;
        Y_o      : out std_logic;
        X_o      : out std_logic );
end exemple;

architecture logique of exemple is
  signal Y_s : std_logic;   --signal interne
begin
--pas de .._o à droite des affectations
  Y_s <= A_i nor B_i;
  Y_o <= Y_s;
  X_o <= Y_s or  B_i;
end logique;
```

Comparaison std_logic_vector ...

- La comparaison se fait en comparant bit à bit les deux vecteurs en commençant par les MSB !
- Exemple de comparaison :

```
signal reg : std_logic_vector(4 downto 0);
signal cnt : std_logic_vector(3 downto 0);
--   reg    >    cnt    résultat
-- "01111" > "0100"   true       correct
-- "01111" > "1000"   false      ERRONÉ
```

Voir : paquetage numeric_std

… compar. sdt_logic_vector et '-' …

- L'exemple ci-dessous n'est pas correct :

```
   signal adresse : std_logic_vector(3 downto 0);
   signal cs : std_logic;
begin
   cs <= '1' when (adresse = "1---") else '0';
     -- Résultat comparaison toujours FAUSSE
```

Le signal cs est toujours à '0' ! Pourquoi ?

… compar. sdt_logic_vector

- Voici la bonne description :
  - CS doit être actif lorsque Adresse = 1- - -,
  - donc lorsque Adresse varie de 1000 à 1111, soit lorsque A(3) = '1'

```
  signal Adresse : std_logic_vector(3 downto 0);
  signal CS : std_logic;

begin
  CS <= '1' when (Adresse(3) = '1') else '0';
  -- Description correcte au cahier des charges
```

Annexes

HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Annexes

Les mots réservés du VHDL93 …

A adapter pour VHDL 2008

abs
access
after
alias
all
and
architecture
array
assert
attribute

begin
block
body
buffer
bus

case
component
configuration
constant

disconnect
downto

else
elsif
end
entity
exit

file
for
function

generate
generic
group
guarded

… mots réservés du VHDL93 …

A adapter pour VHDL 2008

if
impure
in
inertial
inout
is

label
library
linkage
loop

map

mod

nand
new
next
nor
not
null

of
on
open
or

others
out

package
port
postponed
procedure
process
protected
pure

range
record

… mots réservés du VHDL93

A adapter pour VHDL 2008

register
reject
rem
report
return
rol
ror

select
severity
signal
shared
sla

sll
sra
srl
subtype

then
to
transport
type

unaffected
units
until

use
variable

wait
when
while
with

xnor
xor

Historique normes du langage VHDL …

- 1980 : Début du projet financé par le DoD
- 1987 : Standard IEEE Std 1076-1987 (VHDL 87)
- 1993 : Standard IEEE Std 1164-1993 (Std_Logic_1164)
- 1993 : Standard IEEE Std 1076-1993 (VHDL 93)
- 2000 : Standard IEEE Std 1076-2000 (VHDL 2000)
- 2002 : Standard IEEE Std 1076-2002 (VHDL 2002)
- 2008 : Standard IEEE Std 1076-2008 (VHDL 2008)

- Nouvelle version 2019 …

… historique normes du langage VHDL …

- 1995 : Standard IEEE Std 1076.4
  - Vital_Primitive et Vital_Timing pour la simulation après P-R
- 1997 : Standard IEEE Std 1076.3
  - Normalisation des paquetages Numeric_Bit et Numeric_Std
- 1999 : Standard IEEE Std 1076.6
  - Normalisation de la syntaxe pour la synthèse RTL:
    Standard for VHDL Register Transfer Level Synthesis
- 2004 : Standard IEEE Std 1076.6 -2004
  - Evolution de la syntaxe pour la synthèse RTL, pas de changement significatif.

… historique normes du langage VHDL

Autres étapes de l'historique

- 1994 : Approbation ANSI (ANSI/IEEE Std 1076-1993)
- 1996 : Standard IEEE Std 1076.2 (Mathematical Packages)

Autres normalisations

- 1999 : Standard IEEE-1076.1, VHDL-AMS (modélisation mixte)

Normes et méthodologies

- IEEE Std 1800™-2007, Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language
- IEEE Std 1364™-2005, Standard for Verilog® Hardware Description Language
- OVM SystemVerilog User Guide, v 2.0.2, Cadence-Mentor, june 2009

Sites internet PLDs

- Informations des vendeurs d'outils EDA
  http://www.vhdl.org/
- Liste des outils VHDL
  http://www.asic-world.com/vhdl/tools.html
- Site donnant la liste des vendeurs de PLDs (ancien)
  http://www.fpgacentral.com/vendor/directory

Sites internet PLDs

- Divers documentations
  - http://en.wikipedia.org/wiki/Programmable_logic_device
  - http://en.wikipedia.org/wiki/Hardware_description_language
  - http://en.wikipedia.org/wiki/List_of_Verilog_simulators
  - http://www.doulos.com/knowhow
  - http://www.fpga-guide.com/ Il manque dernières générations PLDs

Divers sites

- Informations sur le langage VHDL:
  - http://www.alse-fr.com/tech.php
    - VHDL Quick Reference
    - VHDL Contextual Help for PC (a bit old)
  - http://www.doulos.com/knowhow/vhdl_designers_guide/
  - http://fr.wikipedia.org/wiki/VHDL
  - http://www.asic-world.com/vhdl/tutorial.html

- FPGA Libre: http://fpgalibre.sourceforge.net/intro_en.html

FIN présentation VHDL !

- Questions