



HAL
open science

VHDL 1 : Bases du VHDL et logique combinatoire

Clément Foucher

► **To cite this version:**

Clément Foucher. VHDL 1 : Bases du VHDL et logique combinatoire. Licence. VHDL, IUT Paul Sabatier, Toulouse, France. 2023. hal-04301434

HAL Id: hal-04301434

<https://hal.science/hal-04301434>

Submitted on 23 Nov 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial - ShareAlike 4.0 International License

VHDL

VHSIC Hardware Description Language

BASES DU VHDL ET LOGIQUE COMBINATOIRE

© 2023 Clément Foucher

Cours distribué sous licence libre 

BUT GEII Toulouse S5 ESE

Rappels de 2^{ème} année

NOTIONS FONDAMENTALES

STRUCTURE DU CODE VHDL

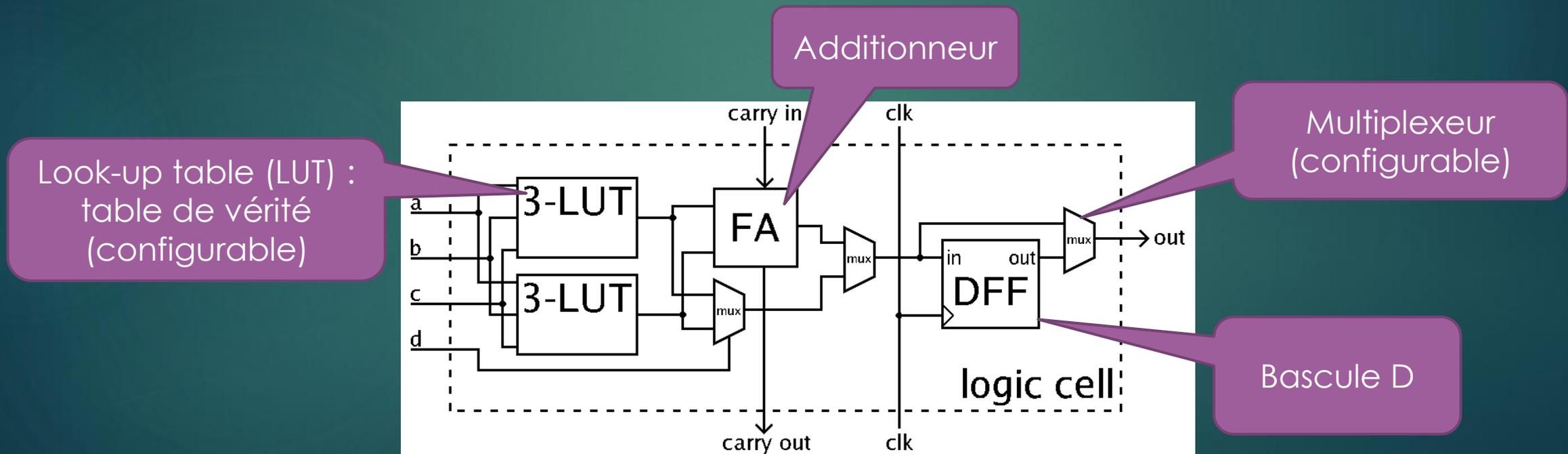
LES TYPES

SYNTAXE DE L'ENTITÉ

SYNTAXE DE L'ARCHITECTURE

Rappels : circuits logiques programmables

- Une cellule programmable de base : Configurable Logic Bloc (CLB)

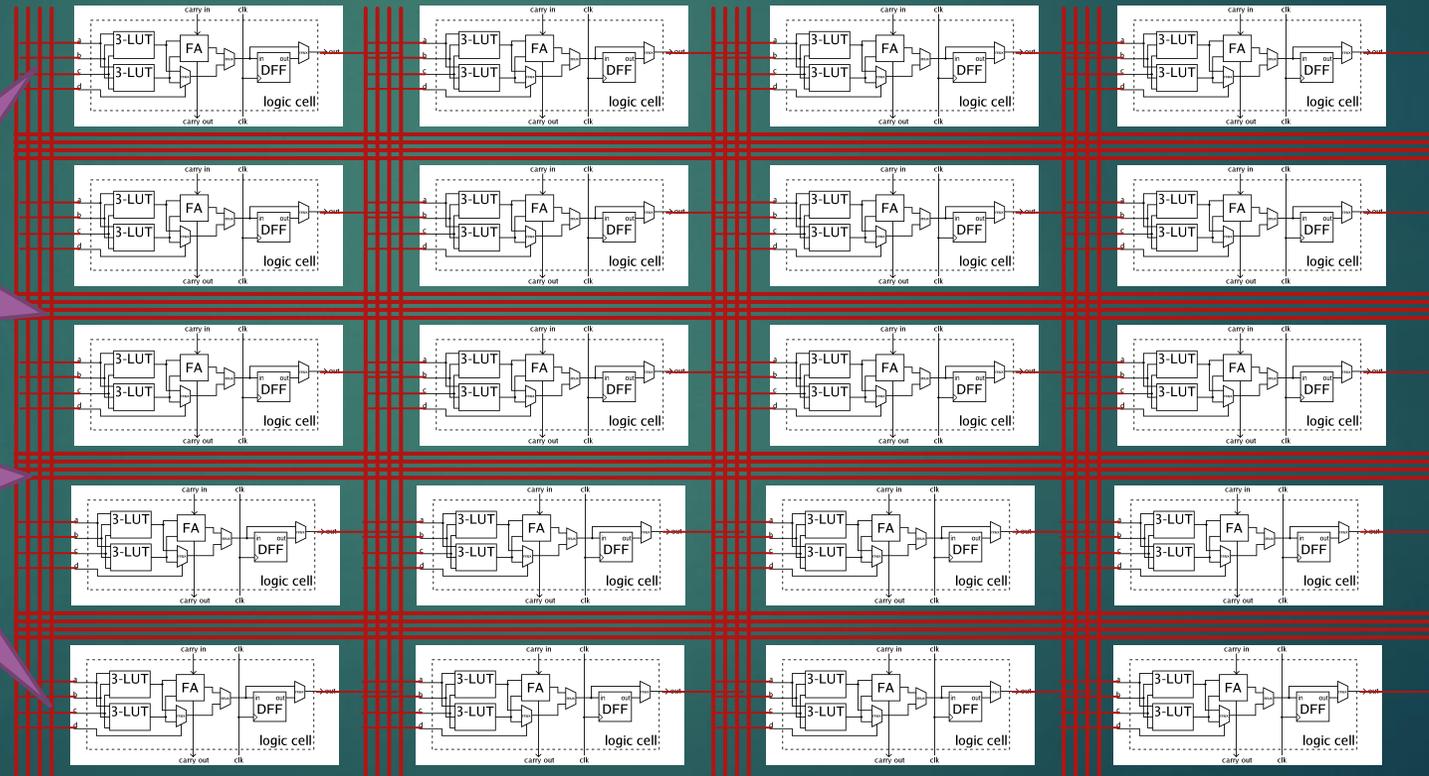


Exemple de cellule logique programmable

Rappels : circuits logiques programmables

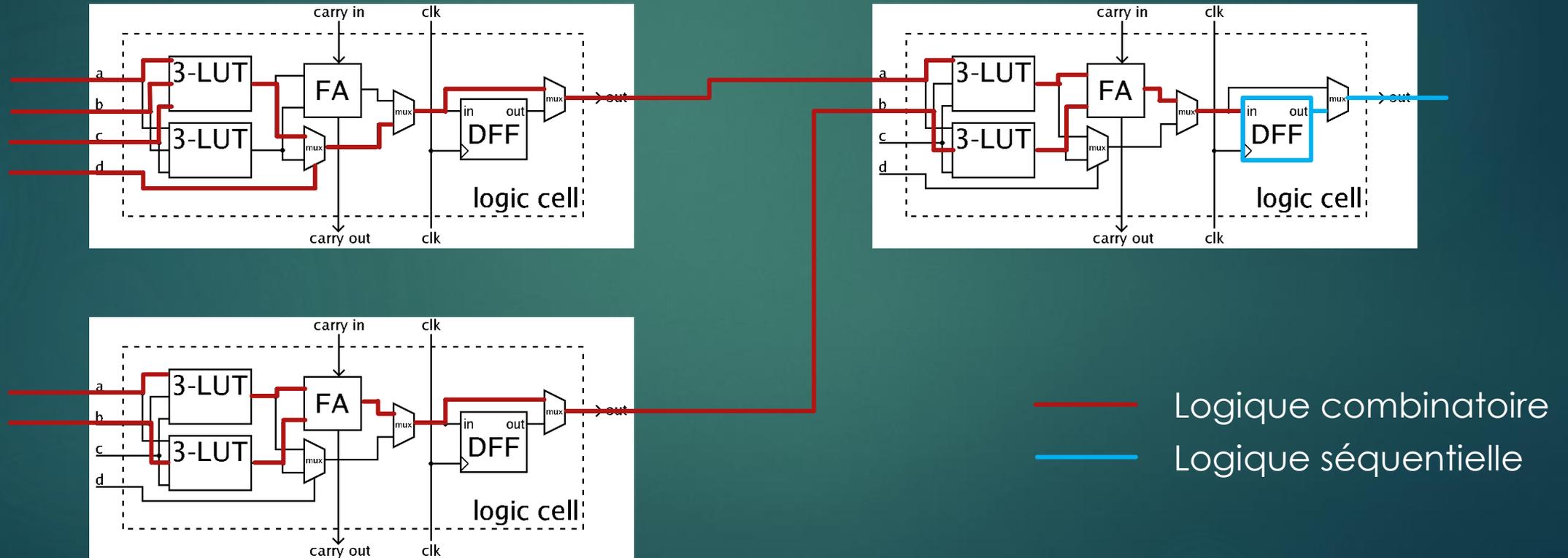
- Plusieurs CLB et un routage configurable

Routage (configurable) : Utilisation de transistors pour choisir quelles lignes sont connectées entre elles



Rappels : circuits logiques programmables

► Exemple de placement / routage



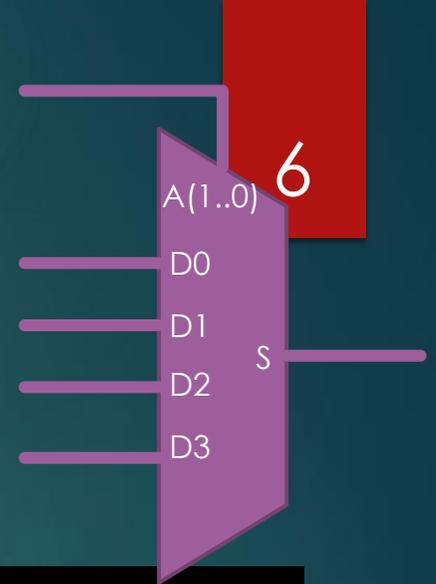
Mais comment coder tout ça ?

- ▶ Des langages de description matériel (Hardware Description Language – HDL) ont été développés
- ▶ Le VHDL et le Verilog sont les deux langages de description les plus répandus
 - ▶ Historiquement, le Verilog s'est plus développé aux États-Unis, et le VHDL en Europe
- ▶ L'année dernière, nous avons utilisé des blocs IP « tout faits »
- ▶ Cette année, nous allons rentrer dans la description des blocs en VHDL

- ▶ Exemple de code en VHDL : multiplexeur 4 vers 1

```
entity Mux_4v1 is
  port(D : in  std_logic_vector(3 downto 0);
        A : in  std_logic_vector(1 downto 0);
        S : out std_logic
        );
end entity;

architecture ar of Mux_4v1 is
begin
  with A select S <=
    D(0) when "00",
    D(1) when "01",
    D(2) when "10",
    D(3) when others;
end architecture;
```



A(1..0)	S
00	D(0)
01	D(1)
10	D(2)
11	D(3)

RAPPELS DE 2^{ÈME} ANNÉE

Notions fondamentales

STRUCTURE DU CODE VHDL

LES TYPES

SYNTAXE DE L'ENTITÉ

SYNTAXE DE L'ARCHITECTURE

Composant

- ▶ En VHDL, on raisonne sur la notion de composant
 - ▶ Analogie avec les composants discrets
 - ▶ Les composants peuvent ensuite être assemblés pour réaliser des composants plus complexes
- ▶ Un composant est caractérisé par
 - ▶ Sa vue externe
 - ▶ Nom
 - ▶ Entrées
 - ▶ Sorties
 - ▶ Son comportement



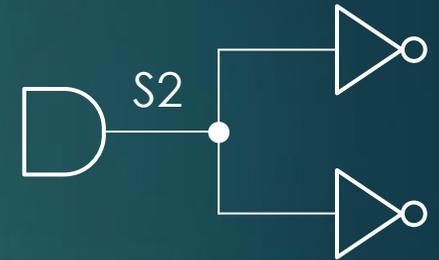
Sera décrite par l'entité (« **entity** »)

Sera décrit par l'**architecture**

Signaux

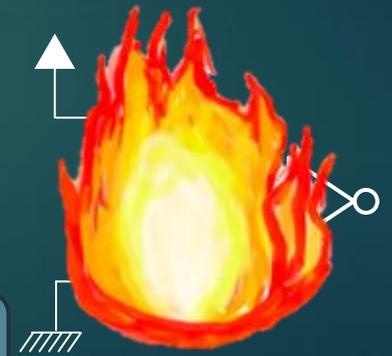
- ▶ En VHDL, on ne raisonne pas sur des variables, mais sur des « signaux »
 - ▶ Mais comme une variable, il doit être nommé et typé
- ▶ Un signal représente un fil (ou un groupe de fils pour les vecteurs)
- ▶ Les entrées et sorties du composant sont des signaux particuliers
 - ▶ La valeur d'une entrée peut être lue, mais pas modifiée
 - ▶ La valeur d'une sortie peut être affectée, mais pas lue
- ▶ La valeur d'un signal peut-être utilisée à plusieurs endroits
 - ▶ Mais ne peut être modifiée qu'à un seul endroit !
 - ▶ Logique : imaginez ce qu'il se passerait dans ce cas ?

Fil = signal S1



Signaux A et B

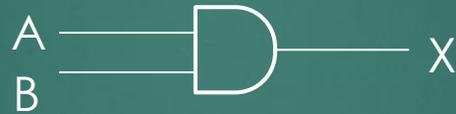
Signaux X et Y



Affectation de signaux

10

- ▶ L'affectation d'un signal se fait de droite à gauche, en utilisant l'opérateur `<=`, et doit se terminer par un `;`
 - ▶ `X <= A and B;`



- ▶ Les signaux à gauche et à droite de l'opérateur d'affectation doivent avoir le même type
 - ▶ Pas de cast implicite en VHDL !

RAPPELS DE 2^{ÈME} ANNÉE
NOTIONS FONDAMENTALES

Structure du code VHDL

LES TYPES

SYNTAXE DE L'ENTITÉ

SYNTAXE DE L'ARCHITECTURE

- ▶ En VHDL, un composant correspond à un fichier
 - ▶ Bonne pratique : le nom du fichier doit correspondre au composant
 - ▶ Exemple : pour un composant nommé « mux4v1 », le fichier se nommera « mux4v1.vhd »
- ▶ Il existe d'autres types de fichiers
 - ▶ Packages
 - ▶ Fonctions
- ▶ On en verra certains dans les prochains cours
- ▶ Contrairement à la plupart des langages, un code VHDL n'est pas sensible à la casse !
 - ▶ `entity = Entity = ENTITY = EnTiTy`

Structure d'un composant

13

▶ Un fichier décrivant un composant est constitué de trois parties

- ▶ Les appels de bibliothèques
- ▶ L'entité (entity)
- ▶ L'architecture

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux_4v1 is
port(D : in  std_logic_vector(3 downto 0);
      A : in  std_logic_vector(1 downto 0);
      S : out std_logic
      );
end entity;

architecture ar of Mux_4v1 is
begin
  with A select S <=
    D(0) when "00",
    D(1) when "01",
    D(2) when "10",
    D(3) when others;
end architecture;
```

Bibliothèques

- ▶ L'appel à une bibliothèque se fait en 2 lignes
 - ▶ Appel à la bibliothèque générale
 - ▶ Mot-clé `library`
 - ▶ Indication de la partie de la bibliothèque que l'on souhaite utiliser
 - ▶ Mot-clé `use`
- ▶ Pour représenter les bits, on utilise le type `std_logic`
 - ▶ Il faudra donc toujours indiquer que l'on souhaite utiliser la bibliothèque `std_logic_1164`
- ▶ Analogie avec le C++
 - ▶ Équivalent du `#include`

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux_4v1 is
port(D : in  std_logic_vector(3 downto 0);
     A : in  std_logic_vector(1 downto 0);
     S : out std_logic
     );
end entity;

architecture ar of Mux_4v1 is
begin
  with A select S <=
    D(0) when "00",
    D(1) when "01",
    D(2) when "10",
    D(3) when others;
end architecture;
```

Entité

15



- ▶ L'entité représente les entrées/sorties du composant
- ▶ On peut l'écrire à partir de la vue externe du composant
 - ▶ À ce niveau-là, on n'a pas besoin de connaître le comportement ni le contenu du composant
- ▶ Analogie avec le C++
 - ▶ Équivalent de l'en-tête d'une fonction

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux_4v1 is
port(D : in  std_logic_vector(3 downto 0);
     A : in  std_logic_vector(1 downto 0);
     S : out std_logic
     );
end entity;

architecture ar of Mux_4v1 is
begin
    with A select S <=
        D(0) when "00",
        D(1) when "01",
        D(2) when "10",
        D(3) when others;
end architecture;
```

Architecture

- ▶ L'architecture contient la description du contenu du composant
- ▶ Il s'agit du code décrivant le comportement que l'on souhaite donner au composant
- ▶ Analogie avec le C++
 - ▶ Équivalent du corps d'une fonction

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux_4v1 is
port(D : in  std_logic_vector(3 downto 0);
      A : in  std_logic_vector(1 downto 0);
      S : out std_logic
      );
end entity;

architecture ar of Mux_4v1 is
begin
  with A select S <=
    D(0) when "00",
    D(1) when "01",
    D(2) when "10",
    D(3) when others;
end architecture;
```

RAPPELS DE 2^{ÈME} ANNÉE

NOTIONS FONDAMENTALES

STRUCTURE DU CODE VHDL

Les types

SYNTAXE DE L'ENTITÉ

SYNTAXE DE L'ARCHITECTURE

Std_logic

18

Rappel : il est nécessaire d'appeler la bibliothèque `std_logic_1164` pour pouvoir utiliser les `std_logic`

- ▶ Un bit (= 1 fil) est représenté par le type `std_logic`
- ▶ La valeur d'un `std_logic` s'indique avec des guillemets simples :
 - ▶ '0'
 - ▶ '1'

Std_logic_vector

19

Rappel : il est nécessaire d'appeler la bibliothèque `std_logic_1164` pour pouvoir utiliser les `std_logic`

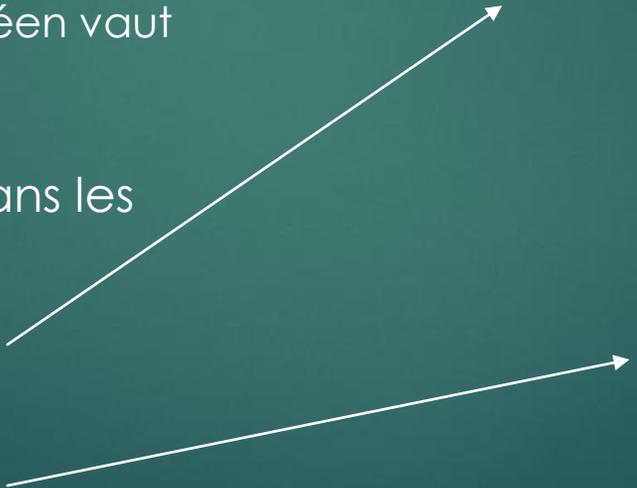
- ▶ Plusieurs bits (groupe de fils) peuvent être représentés par un vecteur de `std_logic` : `std_logic_vector`
- ▶ À la déclaration, il faut indiquer le « range » du vecteur, c'est-à-dire l'indice du poids faible et du poids fort
 - ▶ On placera le poids fort à gauche et le poids faible à droite, avec le mot-clé `downto` :
 - ▶ `signal s : std_logic_vector(7 downto 0)`
 - ▶ Équivalent à la syntaxe que vous connaissez `[7..0]`
 - ▶ Cette syntaxe déclare 8 bits : 7, 6, 5, 4, 3, 2, 1 et 0
- ▶ Pour accéder à un bit particulier du vecteur on utilise les parenthèses :
 - ▶ `s(2)`
 - ▶ Il est possible d'extraire plusieurs bits consécutifs pour obtenir un sous-vecteur
 - ▶ `signal t : std_logic_vector(3 downto 0)`
 - ▶ `t <= s(5 downto 2)`
- ▶ La valeur d'un vecteur s'indique avec des guillemets doubles :
 - ▶ `"01110"`
 - ▶ `"000000000001"`

Booléens

- ▶ On travaille avec des `std_logic`, qui représentent des bits
- ▶ Un booléen, c'est pas la même chose qu'un bit ?
 - ▶ Non !
 - ▶ Un bit vaut 0 ou 1, un booléen vaut vrai ou faux
- ▶ Les booléens sont rarement déclarés, mais sont utiles dans les conditions
- ▶ Par exemple, ce code ne fonctionnera pas
- ▶ Il faut utiliser celui-ci

```
(...  
  
signal s : std_logic;  
  
(...  
  
if s then  
    (...  
end if;
```

```
(...  
  
signal s : std_logic;  
  
(...  
  
if s='1' then  
    (...  
end if;
```



RAPPELS DE 2^{ÈME} ANNÉE

NOTIONS FONDAMENTALES

STRUCTURE DU CODE VHDL

LES TYPES

Syntaxe de l'entité

SYNTAXE DE L'ARCHITECTURE

Entité

22

- ▶ L'entité débute par
`entity <nom> is`
- ▶ Et se termine par
`end entity;`
- ▶ Notez le « ; » qui termine l'instruction `entity`

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux_4v1 is
port(D : in  std_logic_vector(3 downto 0);
     A : in  std_logic_vector(1 downto 0);
     S : out std_logic
     );
end entity;

architecture ar of Mux_4v1 is
begin
    with A select S <=
        D(0) when "00",
        D(1) when "01",
        D(2) when "10",
        D(3) when others;
end architecture;
```

- ▶ L'entité contient uniquement la liste des ports
 - ▶ Introduits par l'instruction `port(<liste des ports>);`
 - ▶ Notez le « ; » qui termine l'instruction `port`
- ▶ La liste des ports est une série de ports
 - ▶ Sous la forme :
`<nom> : <i/o> <type>`
- ▶ Les ports sont *séparés* par des « ; »
 - ▶ Pas de « ; » à la fin de la liste
 - ▶ Utilisation particulière des « ; »
 - ▶ Dans le reste du code, les « ; » *terminent* une instruction
 - ▶ C'est le seul endroit où ils sont utilisés pour *séparer* des instructions

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux_4v1 is
port(D : in  std_logic_vector(3 downto 0);
     A : in  std_logic_vector(1 downto 0);
     S : out std_logic
     );
end entity;

architecture ar of Mux_4v1 is
begin
    with A select S <=
        D(0) when "00",
        D(1) when "01",
        D(2) when "10",
        D(3) when others;
end architecture;
```

Exercice

24

- ▶ Écrire l'entité correspondant à la vue externe ci-dessous



```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux_4v1 is
port(D : in std_logic_vector(3 downto 0);
      A : in std_logic;
      S : in std_logic;
      X : out std_logic;
      Y : out std_logic);
end entity;

architecture arch of Mux_4v1 is
begin
  with A select
    D(0) when "00",
    D(1) when "01",
    D(2) when "10",
    D(3) when others;
end architecture;
```

Erreur courante

25



L'icône de danger indique une erreur courante : ce slide est très important et il faudra s'y référer en cas de doute

- ▶ L'erreur la plus fréquente concerne l'utilisation du « ; »

- ▶ En VHDL, le « ; » indique la fin d'une instruction
- ▶ Deux syntaxes en fonction de l'instruction

```
<instruction>  
    (contenu)  
end <instruction>;
```

Forme longue utilisée par :

- entity
- architecture
- process

- ▶ Ou

```
<instruction> (contenu);
```

Forme utilisée pour les autres instructions

Sera vu dans un prochain cours

- ▶ Mais...

- ▶ Le « ; » sert aussi à séparer les signaux dans l'entité
- ▶ Il y a donc des « ; » *entre* les signaux, mais pas à la fin
- ▶ Pour s'en souvenir, mettre mentalement les signaux sur une seule ligne :
- ▶ `port(A : in std_logic ; X : out std_logic);`

« ; » de séparation

« ; » de fin de l'instruction port

RAPPELS DE 2^{ÈME} ANNÉE

NOTIONS FONDAMENTALES

STRUCTURE DU CODE VHDL

LES TYPES

SYNTAXE DE L'ENTITÉ

Syntaxe de l'architecture

Architecture

27

- ▶ L'architecture débute par
`entity <nom ar> of <nom compo> is`
- ▶ Et se termine par
`end architecture;`
- ▶ Notez le « ; » qui termine l'instruction architecture
- ▶ Le mot-clé `begin` sépare les déclarations de signaux du contenu
 - ▶ Dans cet exemple, il n'y a pas de déclarations
 - ▶ Si il y en avait, il faudrait les placer juste avant le `begin`

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mux_4v1 is
port(D : in  std_logic_vector(3 downto 0);
      A : in  std_logic_vector(1 downto 0);
      S : out std_logic
      );
end entity;

architecture ar of Mux_4v1 is
begin
  with A select S <=
    D(0) when "00",
    D(1) when "01",
    D(2) when "10",
    D(3) when others;
end architecture;
```

Déclaration de signaux internes

28

- ▶ Les signaux internes ne sont ni des entrées ni des sorties, et sont inaccessibles de l'extérieur du composant
- ▶ Ils doivent être déclarés dans l'architecture, avant le begin
- ▶ La syntaxe est :
 - ▶ `signal <nom> : <type>;`

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mon_compo is
port(A : in  std_logic;
      B : in  std_logic;
      X : out std_logic
      );
end entity;

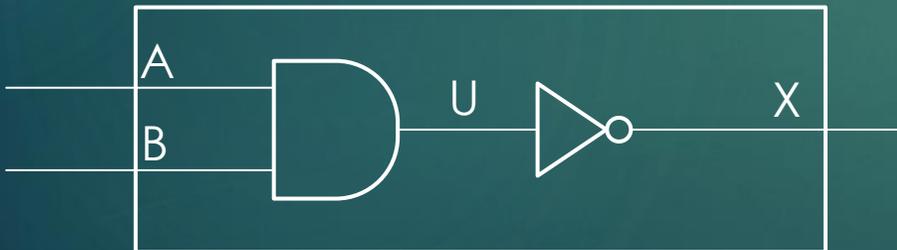
architecture ar of Mon_compo is
    signal U : std_logic;
begin

end architecture;
```

Exercice

29

- ▶ Ordre dans lequel les instructions sont exécutées
 - ▶ On vous a bassinés en informatique et automatisme avec l'ordre d'exécution des instructions
 - ▶ Qui peut deviner l'ordre dans lequel les affectations de U et X sont réalisées ?



```
library IEEE;
use IEEE.std_logic_1164.all;

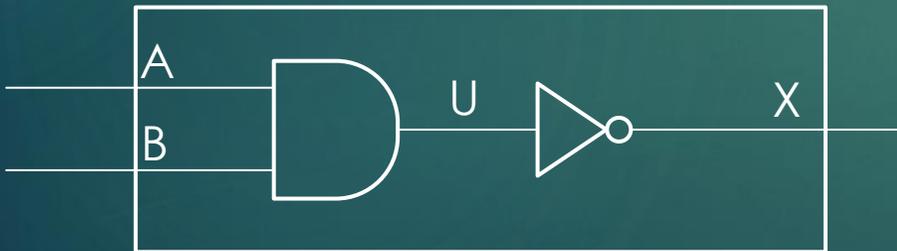
entity Mon_compo is
port(A : in  std_logic;
      B : in  std_logic;
      X : out std_logic
    );
end entity;

architecture ar of Mon_compo is
    signal U : std_logic;
begin
    U <= A and B;
    X <= not U;
end architecture;
```

Exercice

30

- ▶ Le code ci-contre décrit exactement le même composant que le précédent !



```
library IEEE;
use IEEE.std_logic_1164.all;

entity Mon_compo is
port(A : in  std_logic;
      B : in  std_logic;
      X : out std_logic
    );
end entity;

architecture ar of Mon_compo is
    signal U : std_logic;
begin
    X <= not U;
    U <= A and B;
end architecture;
```

Opérateurs logiques de base

- ▶ Les opérateurs logiques suivants existent pour les `std_logic` (et pour les vecteurs) sont les suivants :
 - ▶ `and`
 - ▶ `nand`
 - ▶ `or`
 - ▶ `nor`
 - ▶ `xor`
 - ▶ `xnor`
 - ▶ `not`
- ▶ Toutefois, leur utilisation est généralement limitée : on préférera des opérateurs plus puissants comme ceux que l'on va voir maintenant, ou l'utilisation de `process` que l'on verra dans un cours ultérieur

Opérateur `when ... else`

32

- ▶ L'opérateur `when ... else` permet de choisir une valeur à retourner en fonction d'une condition
- ▶ Syntaxe :
 - ▶ `X when C else Y`
 - ▶ La valeur retournée sera X si C est vrai, sinon Y sera retourné
 - ▶ Cette valeur est en général utilisée pour affecter un signal
- ▶ Exemple
 - ▶ `A <= '1' when B = "101" else '0';`
 - ▶ Ici, A prendra la valeur 1 si B vaut 5, sinon A prendra la valeur 0

Notez que l'opérateur d'égalité est `=`, comme en C
Il retourne un booléen

Opérateur when ... else

33

- ▶ Mais il est possible de chaîner les opérateurs when ... else pour tester plusieurs conditions :
 - ▶ Dans l'exemple `X when C else Y`, Y peut lui-même être un when ... else
 - ▶ Exemple :
 - ▶ `A <= "00" when B = "101" else "01" when C = '1' else "11";`
 - ▶ Ici, a prendra 0 si B vaut 5, mais si B ne vaut pas 5, alors on teste si C vaut 1
- ▶ Lorsqu'il y a plusieurs conditions, on préférera les présenter sur plusieurs lignes et aligner les mots-clés pour une meilleure lisibilité :

```
A <= "00" when B = "101" else
    "01" when B = "111" else
    "10" when C = "11" else
    "11";
```

Opérateur `with ... select`

34

- ▶ Autre opérateur très utilisé : le `with ... select` permet d'affecter une valeur à un signal en fonction d'une autre valeur
- ▶ Syntaxe :

```
with B select A <= (...) when (...),  
              (...) when (...),  
              (...) when (...),  
              (...) when others;
```

Valeur à donner à A

En fonction de la valeur de B

- ▶ On terminera *toujours* par `others`, même si on a traité toutes les valeurs possibles de B

▶ Pour comprendre ce point, se référer au « bonus » en fin de cours : on n'a en réalité *jamais* traité toutes les valeurs possibles d'un `std_logic`

Opérateur `with ... select`

35

▶ Exemple

```
with B select A <= '0' when "00",  
              '1' when "01",  
              '1' when "10",  
              '0' when others;
```

Valeur à donner à A

En fonction de la valeur de B

Différence entre `with ... select` et `when ... else`

36

- ▶ `with ... select` renvoie une valeur différente en fonction de la valeur d'UN signal
 - ▶ Représente un multiplexeur, ou une table de vérité simple
- ▶ `when ... else` permet d'indiquer une condition arbitraire pour chaque valeur
 - ▶ Permet notamment de représenter les priorités entre différents signaux

Exercice

- ▶ Quel est l'opérateur le plus adapté pour représenter la table de vérité ci-contre ?
 - ▶ `with ... select`
- ▶ Décrire le code correspondant à son implémentation

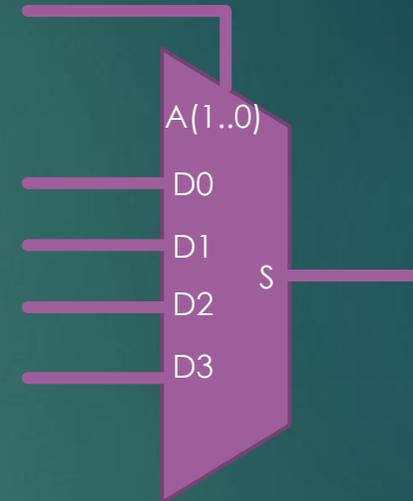
A(1..0)	S
00	D(0)
01	D(1)
10	D(2)
11	D(3)

```
with A select S <=  
  D(0) when "00",  
  D(1) when "01",  
  D(2) when "10",  
  D(3) when others;
```

Exercices

38

- ▶ Quel est l'opérateur le plus adapté pour représenter le composant ci-contre ?
 - ▶ `with ... select`
- ▶ Décrire le code correspondant à son implémentation



```
with A select S <=  
  D(0) when "00",  
  D(1) when "01",  
  D(2) when "10",  
  D(3) when others;
```

Exercice

- ▶ Donner la table de vérité du code ci-contre

```
with A select X <= '0' when "00",  
              '1' when "01",  
              '1' when "10",  
              '0' when others;
```

A	X
00	0
01	1
10	1
11	0

Exercice

40

- ▶ Quelle instruction est la plus adaptée pour implémenter la table de vérité ci-contre ?
 - ▶ Ici, bien que A et B soit deux signaux indépendants, on peut les traiter comme un vecteur
 - ▶ On utilise l'opérateur de concaténation & pour former un vecteur
 - ▶ On peut donc utiliser `with ... select`, qui est plus simple

A	B	X
0	0	00
0	1	01
1	0	11
1	1	10

AB	X
00	00
01	01
10	11
11	10

```
signal AB : std_logic_vector(1 downto 0);  
  
(...)  
  
AB <= A&B;  
  
with AB select X <= '0' when "00",  
                  '1' when "01",  
                  '1' when "10",  
                  '0' when others;
```

Exercice

41

- ▶ Quel est l'opérateur le plus adapté pour représenter la table de vérité ci-contre ?
 - ▶ when ... else
- ▶ Décrire le code correspondant à son implémentation

en	A	B	S
0	*	*	0
1	0	*	1
1	1	0	X
1	1	1	Y

```
S <= '0' when en = '0',  
      '1' when A = '0',  
      X  when B = '0',  
      Y  when others;
```

Cas particulier

- ▶ Attention, lorsque plusieurs signaux ont la même priorité, il faut les répéter à chaque ligne
 - ▶ Ici, en est prioritaire sur les autres signaux, mais A et B ont la même priorité

en	A	B	S
0	*	*	0
1	0	0	X
1	0	1	Y
1	1	0	Z
1	1	1	T

```
S <= '0' when en = '0',  
      X when A = '0' and B = '0',  
      Y when A = '0' and B = '1',  
      Z when A = '1' and B = '0',  
      T when others;
```

43



L'icône d'information indique que le contenu de ces slides est à titre informatif, il ne vous est pas demandé de le retenir

Bonus

Variantes de syntaxe

44



- ▶ Il existe plusieurs manières de terminer une entité

- ▶ `end entity;`
- ▶ `end <nom de l'entité>;`
- ▶ `end entity <nom de l'entité>;`

- ▶ Il existe plusieurs manières de terminer une architecture

- ▶ `end architecture;`
- ▶ `end <nom de l'archi>;`
- ▶ `end architecture <nom de l'archi>;`

Évolution du VHDL

45



- ▶ Citation d'un slide précédent : « La valeur d'une sortie peut être affectée, mais pas lue »
 - ▶ Ceci n'est plus tout à fait vrai : depuis le VHDL 2008, il est possible de lire la valeur d'une sortie
 - ▶ Il s'agit d'une simplification destinée à éviter la déclaration d'un signal intermédiaire lorsque l'on a besoin de réutiliser la valeur fournie en sortie
 - ▶ Toutefois, plus de 10 ans après, le VHDL 2008 est *très loin* d'être totalement pris en charge par tous les outils de développement
 - ▶ Ainsi, cette année, nous raisonnerons sur le fait qu'une sortie ne *peut pas* être lue, mais il est possible qu'en entreprise, vous utilisiez du VHDL 2008 : dans ce cas, ne soyez pas surpris si vous tombez sur une lecture de sortie

Les types : std_logic

46



- ▶ Un std_logic peut prendre les valeurs '0' et '1', mais également :
 - ▶ 'H', 'L' (high et low)
 - ▶ 'Z' (haute impédance)
 - ▶ 'W' (weak)
 - ▶ 'U' (non initialisé)
 - ▶ 'X' (valeur inconnue)
 - ▶ '-' (valeur sans importance)
- ▶ Vecteurs de std_logic
 - ▶ Si, dans l'immense majorité des cas, on place le poids fort à gauche et le poids faible à droite (ex: 7 downto 0), il est également possible de faire l'inverse (0 to 7)
 - ▶ Cela peut être utile dans certains cas très particuliers qui ne nous intéresseront pas cette année, par exemple pour inverser simplement l'ordre des bits entre deux vecteurs
 - ▶ Il est également possible de déclarer des vecteurs dont les indices ne vont pas jusqu'à 0 : (6 downto 3) OU (5 to 15)

Les types : quelques autres types

47



▶ integer

- ▶ Accessibles sans appel de bibliothèque
- ▶ Utiles pour réaliser des compteurs, choisir un indice dans un vecteur, etc.
- ▶ Attention, pour pouvoir être synthétisés, il est nécessaire de leur donner un range (un minimum et un maximum), qui permettra au compilateur de connaître le nombre de bits nécessaires pour les représenter

▶ signed et unsigned

- ▶ Accessibles dans la bibliothèque `IEEE.numeric_std`
- ▶ Vecteurs de bits sur lesquels il est possible de réaliser les opérations arithmétiques en tenant compte ou non d'un bit de signe

Pour aller plus loin

48



- ▶ Quelques références
 - ▶ [Les types](#)
 - ▶ [VHDL 2008](#)
 - ▶ [Signed et unsigned](#)