

C2 – Structure d'une description VHDL

Yann DOUZE
VHDL

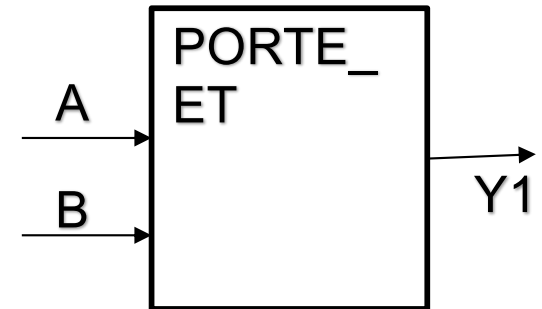


Structure d'une description VHDL

- Une description **VHDL** est composée de 2 parties **indissociables** à savoir :
 - **L'entité** (**ENTITY**), elle définit les entrées et sorties.
 - **L'architecture** (**ARCHITECTURE**), elle contient les instructions **VHDL** permettant de réaliser le fonctionnement attendu.

Exemple ET Logique (AND)

```
entity PORTE_ET is  
  port (A,B :in std_logic;  
        Y1:out std_logic);  
end entity;
```



```
architecture DESCRIPTION of PORTE_ET is  
begin  
  Y1 <= A and B;  
end architecture;
```



Déclaration des bibliothèques

- Toute description **VHDL** utilisée pour la synthèse a besoin de bibliothèques.
- L'**IEEE** (Institut of **E**lectrical and **E**lectronics **E**ngineers) les a normalisées et plus particulièrement la bibliothèque **IEEE 1164**.
- Elles contiennent les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques,...

```
Library ieee;
```

```
Use ieee.std_logic_1164.all;
```

```
Use ieee.numeric_std.all;
```



Illustration

```
library ieee;  
Use ieee.std_logic_1164.all;  
Use ieee.numeric_std.all;
```

Déclaration des bibliothèques

```
-- décodeur  
-- Un parmi quatre
```

Commentaires, en VHDL ils commencent par - -




Déclaration de l'entité

- **Syntaxe:**

```
entity NOM_DE_L_ENTITE is  
    port ( Description des signaux d'entrées /sorties ... );  
end entity;
```

- Exemple :

```
entity SEQUENCEMENT is  
    port (  
        CLOCK : in std_logic;  
        RESET : in std_logic;  
        Q : out std_logic_vector(1 downto 0)  
    );  
end entity;
```



- **Remarque :** Après la dernière définition de signal de l'instruction `port` il ne faut jamais mettre de point virgule.

Déclaration des signaux E/S

- L'instruction **port** :

Syntaxe: *NOM_DU_SIGNAL* : *sens* *type*;

Exemple: *CLOCK* : *in* *std_logic*;

BUS : *out* *std_logic_vector* (7 *downto* 0);

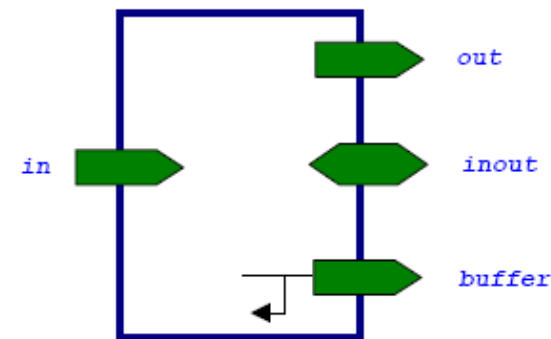
- *Le sens du signal* :

in : pour un signal en entrée.

out : pour un signal en sortie.

inout : pour un signal en entrée sortie

buffer : pour un signal en sortie mais pouvant être lu (déconseillé).





Le Type

Le **TYPE** utilisé pour les signaux d'entrées / sorties est :

- le **std_logic** pour un signal.
- le **std_logic_vector** pour un bus composé de plusieurs signaux.

Par exemple un bus bidirectionnel de 5 bits s'écrira :

LATCH : **inout std_logic_vector (4 downto 0)** ;

Où **LATCH(4)** correspond au **MSB** et **LATCH(0)** correspond au **LSB**.

MSB : Most Significant Bit

LSB : Less Significant Bit

Les valeurs que peuvent prendre un signal de type **std_logic** sont :

- **'0'** ou **'L'** : pour un niveau bas.
- **'1'** ou **'H'** : pour un niveau haut.
- **'X'** ou **'W'** : pour un niveau inconnu.
- **'U'** : pour non initialisé.
- **'Z'** : pour état haute impédance.
- **'-'** : Quelconque, c'est à dire n'importe quelle valeur. ('0' ou '1')



Description de l'architecture

- L'architecture est relative à une entité. Elle décrit le corps du design, son comportement, elle établit à travers les instructions les relations entre les entrées et les sorties.

- **Exemple :**

-- Opérateurs logiques de base

```
entity PORTES is
    port (A,B :in std_logic;
          Y1,Y2,Y3,Y4,Y5,Y6,Y7:out std_logic);
end entity;
architecture DESCRIPTION of PORTES is
begin
    Y1 <= A and B;
    Y2 <= A or B;
    Y3 <= A xor B;
    Y4 <= not A;
    Y5 <= A nand B;
    Y6 <= A nor B;
    Y7 <= not(A xor B);
end architecture;
```



VHDL : langage concurrent ?

```
architecture DESCRIPTION of DECOD is
Begin
    -- instructions concurrentes
    D0 <= (not(IN1) and not(IN0)); -- première instruction
    D1 <= (not(IN1) and IN0);      -- deuxième instruction
end architecture;
```

- Entre le BEGIN et le END de l'architecture, on est dans un contexte d'instructions concurrentes.
- Instructions concurrentes :
 - L'ordre dans lequel sont écrites les instructions n'a aucune importance.
 - Toutes les instructions sont évaluées et affectent les signaux de sortie en même temps.
 - C'est la différence majeure avec un langage informatique.

L'architecture ci dessous est équivalente :

```
architecture DESCRIPTION of DECOD is
begin
    D1 <= (not(IN1) and IN0);          -- deuxième instruction
    D0 <= (not(IN1) AND not(IN0));    -- première instruction
end architecture;
```



Clause de contexte implicite

```
library STD, WORK;  
use STD.STANDARD.all;
```

Inclut par défaut

```
INTEGER    0  1  99  
BIT         '0'  '1'  
BOOLEAN    FALSE TRUE  
STRING     "Hello World"  
TIME       10 NS
```

Types du package
STD.STANDARD



Noms des labels

Identifiant = lettres, nombres et underscores

```
G4X6      Gate_45      \extended!`£$%^&*() \
TheState   The_State
```

Noms différents

La casse est ignorée pour les identifiants

```
INPUT      Input      input
```

mêmes noms

Beaucoup d'identifiant sont des mots réservés

```
And  buffer  bus  function  register  select
```

Identifiants illégaux ...

```
4plus4  A$1  V-3  The__State  _State_
```

Double underscore



Recommandations

- Donner le même nom au fichier que l'entité.
 - Exemple : si l'entité s'appelle **counter**, nommé le fichier **counter.vhd**
- Lorsque vous utilisez un signal en logique négative, rajouter **_n** à la fin du nom du signal.
 - Exemple : si vous utilisez un **reset** actif à l'état bas, nommez le **reset_n**