# Guide du VHDL

## Table des matières


Types de signaux ..... 2
Conversion de types ..... 2
Valeurs du std_logic ..... 3
Portes logiques ..... 3
- Porte NOT ..... 3
- Porte AND ..... 3
- Porte OR ..... 3
- Porte XOR ..... 3

Déclarations ..... 4
- Déclaration d'une entité ..... 4
- Déclaration d'un signal interne ..... 4
- Déclaration des signaux suivant leur type ..... 4
- Déclaration d'un composant ..... 5

Instanciation ..... 5
Instructions concurrentes ..... 5
- Affectation ..... 5
- Condition ..... 5
- Sélection ..... 6
- Boucle par itération ..... 6

Instructions séquentielles ..... 6
- Affectation ..... 6
- Condition ..... 6
- Sélection ..... 6
- Boucle par iteration ..... 7
- Boucle ..... 7

## Types de signaux

`std_logic` : valeur binaire sur 1 bit

`std_logic_vector` : valeur binaire sur plusieurs bits

`unsigned` : valeur binaire en code binaire naturel (sous-type de std_logic_vector)

`signed` : valeur binaire en code complément à 2 (le MSB est un bit de signe sous-type de `std_logic_vector`)

`boolean` : soit *true* soit *false* (Valeur résultat d'une condition, exemple : la condition A=5 vaut *true* ou *false)*

`positive` : valeur décimale (entier natuurel >0)

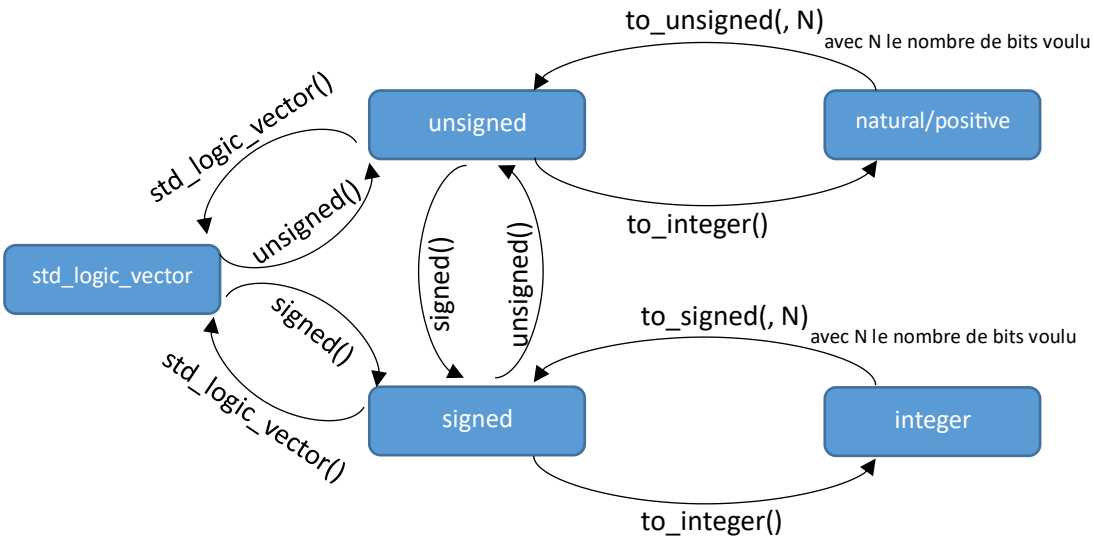
`natural` : valeur décimale (entier naturel)

`integer`: valeur décimale (entier)

`real` : valeur réelle (à n'utiliser que pour les constantes ou les paramètres génériques ou dans un testbench) Exemples de réels :

```
constante c1 : real := 0.0 ;
constante c2 : real := 1.0E-9 ;
```

## Conversion de types



## Valeurs du std_logic

| VHDL | Signification | Explication |
|---|---|---|
| ‘U’ | Undefined | Non défini (signal non afecté) |
| ‘X’ | Unknown | Conflit |
| ‘0’ | 0 | |
| ‘1’ | 1 | |
| ‘Z’ | High impedance | Un Z qui veut dire impédance (souvent noté Z) |
| ‘W’ | Weak unknown | Conflit entre les 2 valeurs suivantes |
| ‘L’ | Low : 0 faible | Utile pour décrire un pull-down |
| ‘H’ | Hogh : 1 fable | Utile pour décrire un pull-up |
| ‘-‘ | Don’t care | Peu importe la valeur |

Une valeur sur 1 bit est spécifiée entre guillemets simples :

```
A <= '1' ;
```

Une valeur sur plusieurs bits est spécifiée entre guillemets doubles :

```
A <= "00100111" ;
```
Valeur donnée en binaire

```
A <= X"3F" ;
```
Valeur donnée en hexadécimal

## Portes logiques

### Porte NOT

```
signal_out <= NOT signal_in ;
```

### Porte AND

```
signal_out <= signal1 AND signal2 ;
```

### Porte OR

```
signal_out <= signal1 OR signal2 ;
```

### Porte XOR

```
signal_out <= signal1 XOR signal2 ;
```

# Déclarations

## Déclaration d'une entité

Une entité est une description de l'interface d'un composant électronique, qui spécifie les ports d'entrée et de sortie utilisés dans le composant.

optionnel

```
entity entity_name is
  generic (
    parameter1 : type_signal := default_value ;
    parameter2 : type_signal := default_value
  );
  port (
    port_name1 : in  type_signal;
    port_name2 : out type_signal;
    port_name3 : out type_signal
  );
end entity_name;
```

Valeur par défaut du paramètre générique (s'il n'est pas spécifié lors d'une future instanciation)

## Déclaration d'un signal interne

```
signal signal_name : type_signal := intial_value;
```

optionnel

## Déclaration des signaux suivant leur type

```
constant x : positive := 16;
```

std_logic : signal sur 1 bit

```
signal clk : std_logic;
```

std_logic_vector : signal sur plusieurs bits

```
signal A : std_logic_vector (7 downto 0);
signal B : std_logic_vector (2 to 6);
```

unsigned et signed : bus représentant une donnée arithmétique

```
signal U : unsigned (7 downto 0);
signal S : signed (x-1 downto 0);
```

integer : entiers (sur 32 bits)

```
signal I : integer range -4 to 7;
```

natural : entiers positifs (sur 31 bits)

```
signal N : natural range 0 to x; -- sera codé sur 5 bits
```

positive: entiers strictement positifs (sur 31 bits)

```
signal P : positive range 8 to 15; -- sera codé sur 3 bits
```

types énumérés : types personnalisé, très utile pour décrire des machines à états

```
type state is (attente, init, marche, marchepas);
signal current_state : state;
```

### Déclaration d'un composant

Un composant est une entité (entity) externe que l'on souhaite utiliser dans un autre module ou architecture. Le composant permet de déclarer les entrées et sorties de l'entité, ainsi que ses caractéristiques (les paramètres génériques).

optionnel

```
component entity_name is
  generic (
    parameter1 : type_signal := default_value ;
    parameter2 : type_signal := default_value
  );
  port (
    port_name1 : in  type_signal;
    port_name2 : out type_signal;
    port_name3 : out type_signal
  );
end component;
```

## Instanciation

Permet d'utiliser un code VHDL déjà développé (et validé) dans un autre code (sans faire de copier-coller qui pourrait introduire des erreurs). Cela permet alors d'utiliser une entité déjà créée à l'intérieur d'une autre.

optionnel

```
instance_name : entity_name
  generic map (
    parameter1 => 10,
    parameter2 => 100.0E6
  )
  port map(
    port_name1 => signal1
    port_name2 => signal2,
    port_name3 => signal3
  );
end component;
```

A gauche, noms des entrées et sorties de l'entité

A droite, noms des signaux de l'architecture actuelle

## Instructions concurrentes

### Affectation

```
signal1 <= expression1 ;
```

### Condition

```
signal1 <= expression1 when condition1
      else expression2 when condition2
      -- (…)
      else expresion3 ;
```

### Sélection

```
with signal1 select
      signal2 <= expression1 when valeur1,
            expression2 when valeur2,
            -- (…)
            expressionX when others;
```

Signifie : dans tous les autres cas

La valeur du *signal2* est piloté par le *signal1*. Le *signal2* vaudra *expression1* lorsque le *signal1* vaudra *valeur1*, *expression2* s'il vaut *valeur2* etc.

### Boucle par itération

Permet de nommer la boucle

```
label1 : for i in 0 to 15
generate
      -- instructions concurrentes
end generate;
```

Pour i allant de 0 à 15 (inclus), faire les instructions entre *generate* et *end generate*. A la fin de chaque itération i prend +1.

*Remarque : l'indice (ici i) n'a pas à être déclaré au préalable.*

## Instructions séquentielles

### Affectation

```
signal1 <= expression1 ;
```

### Condition

```
if condition1 then
       -- instructions séquentielles
elsif condition2 then
      -- instructions séquentielles
--(elsif …)
else
      -- instructions séquentielles
end if;
```

### Sélection

```
case signal1 is
      when valeur1 => -- instructions séquentielles
      when valeur2 => -- instructions séquentielles
      -- (…) when others => -- instructions séquentielles
      end case;
```

Signifie : dans tous les autres cas

## Boucle par iteration

```
for i in 0 to 15 loop
      -- instructions séquentielles
end loop;
```

Pour i allant de 0 à 15 (inclus), faire les instructions entre *loop* et *end loop*. A la fin de chaque itération i prend +1.

*Remarque : l'indice (ici i) n'a pas à être déclaré au préalable.*

## Boucle

```
while condition loop
      -- instructions séquentielles
end loop;
```

Tant que la condition est respetée, faire les instructions entre *loop* et *end loop*.