Représentation des nombres en VHDL

Types de données :

- prédéfinis,
- définis par l'utilisateur

Opérateurs :

- logique,
- relationnel,
- de décalage,
- de concaténation

Représentation des nombres en VHDL

Il existe deux types de données scalaires permettant de représenter des nombres en VHDL :

· Les entiers (qui sont synthétisables) avec par exemple :

signal I : integer range -128 to 127; I sera codé sur 8 bits en CA2

signal J : integer range 0 to 15; J sera codé sur 4 bits non signés

signal K : integer range -32768 to 32767; K sera codé sur 16 bits en CA2

signal L : integer; L sera codé sur 32 bits en CA2 (par défaut)

· Les réels (qui ne sont pas synthétisables mais que l'on peut utiliser dans un testbench) avec

par exemple :

signal X : real; X peut être compris entre -1038 et 1038 (float 32 bits par défaut)

N.B: CA2 signifie Complément à2

On peut aussi utiliser des tableaux de variables de type bit comme bit_vector ou de type std_logic comme std_logic_vector, ce dernier étant le plus utilisé.

Par défaut, ils sont non typés et ne permettent donc pas les opérations arithmétiques telles que l'addition ou la multiplication.

Pour pouvoir faire de l'arithmétique notamment avec le type std_logic_vector, on peut utiliser des packages de la librairie IEEE.

Types de données

VHDL inclut plusieurs types prédéfinis et d'autres qui sont définis dans des packages normalisés.

L'utilisateur peut aussi définir d'autres types.

catégorie	type ou sous-type	source de la définition	valeurs
scalaires	boolean	type prédéfini	FALSE et TRUE
	bit	type prédéfini	'0' et '1'
	character	type prédéfini	256 caractères de la norme ISO 8859-1, avec des abréviations reconnues et certaines qui sont propres à VHDL Les 128 premiers sont les caractères ASCII.
	integer	type prédéfini	plage minimale de $-2^{31}+1$ à $2^{31}-1$
	natural	sous-type prédéfini	0 à $2^{31}-1$
	positive	sous-type prédéfini	1 à $2^{31}-1$
	real	type prédéfini	typiquement −1.7014111E±308 à 1.7014111E±308
	std_logic	Package std_logic_1164	'U' : valeur inconnue, pas initialisée 'X' : valeur inconnue forcée '0' : 0 forcé '1' : 1 forcé 'Z' : haute impédance (pas connecté) 'W' : inconnu faible 'L' : 0 faible 'H' : 1 faible '-' : peu importe (don't care)
composés	bit_vector	type prédéfini	tableau de bit
	string	type prédéfini	tableau de character
	std_logic_vector	Package std_logic_1164	tableau de std_logic
	unsigned	Package numeric_std	tableau de std_logic, interprété comme un nombre binaire non signé
	signed	Package numeric_std	tableau de std_logic, interprété comme un nombre binaire signé en complément à deux

Définition de nouveaux types en VHDL

- Les définitions de types se font dans la partie déclarative de l'architecture.
- On peut définir de nouveaux types avec le mot clé **type** , et des sous-types avec le mot-clé **subtype.**
- On peut aussi définir des types composés dont les éléments:
  - sont tous du même type et sont numérotés (vecteurs ou matrices, **array**)
  - Sont de types différents, appelés structures (**record**)

Exemple:

Nombre d'éléments spécifié

Nombre d'éléments n'est pas spécifié

- type entiers_inf_20 is range 0 to 19;
- type matrice_relle_1_par_10 is array (1 to 10) of real;
- type tableau is **array (natural range <>)** of std_logic_vector (2 downto 0);
- constant vecteur: tableau:= (''000'', ''001'', ''010'', ''011'', ''100'', ''101'', ''110'', ''111'');
- Type nom_de_mois is (janvier, février, mars, avril, mai, juin, juillet, aout, septembre, octobre, novembre, décembre);
- type date is record
  jour : integer range 1 to 31;
  mois: nom_de_mois;
  An : positive range 1 to 3000;
  end record;

- Constant indépendance: date :=(5, juillet, 1962);

Les opérateurs

- **Opérateur arithmétiques : +, -, ***

Chaque opérateur peut porter sur un type signed, unsigned ou integer mais en aucun cas sur un std_logic_vector.

Exemples :

```
signal u1, u2 : unsigned (3 downto 0);
signal s1, s3 : signed (3 downto 0);
signal s2 : signed (4 downto 0);
signal v1 : std_logic_vector (3 downto 0);
signal v2 : std_logic_vector (4 downto 0);
signal i1, i2 : integer;
u1 <= "1001"; -- = 9
s1 <= "1001"; -- = -7
i1 <= 16;
s2 <= u1 + s1; -- = 2
v2 <= u1 + s1; -- = "00010"
s3 <= -s1; -- = 7
```

Opérateurs de comparaison : <, <=, >, >=, =, /=

Chaque opérateur peut porter sur un type signed, unsigned ou integer mais en aucun cas sur un std_logic_vector .

Il retourne un booléen (vrai/faux).

Les comparaisons de comparaison sont:

< inférieur	> supérieur	|= Egal à
<= inférieur ou égal	>= supérieur ou égal	|/= Différent de

Exemples:

```
signal u1, u2 : unsigned (3 downto 0);
signal s1 : signed (3 downto 0);
signal o1,o2 : std_logic;
u1 <= "1001"; -- = 9
u2 <= "0111"; -- = 7
s1 <= "1001"; -- = -7
…
if u1 > u2 then
o1 <= '1'; -- o1 set to '1'
else
o1 <= '0';
end if;
…
if s1 > u2 then
o2 <= '1';
else
o2 <= '0'; -- o2 set to '0'
end if;
```

Fonctions de décalage : SHL, SHR

La fonction de décalage à gauche SHL (SHift Left) et à droite SHR (SHift Right) portent sur un type signed ou unsigned.
Lorsque SHR porte sur un type signed, il réalise l'extension de signe.

```
function shl(arg: unsigned; count: unsigned) return unsigned;

function shl(arg: signed; count: unsigned) return signed;

function shr(arg: unsigned; count: unsigned) return unsigned;

function shr(arg: signed; count: unsigned) return signed;
```

```
architecture comporte of dec is
signal u1, u2, u3 : unsigned(3 downto 0);
signal s1, s2, s3 : signed(3 downto 0);
begin
u1 <= "0010";
u2 <= shl(u1, "10"); -- "1000"
u3 <= shr(u2, "10"); -- "0010" : u2 non signé, pas d'extension de signe
s1 <= "0010";
s2 <= shl(s1, "10"); -- "1000"
s3 <= shr(s2, "10"); -- "1110" : u2 signé, extension de signe
end comporte ;
```

L'opérateur de concaténation:

L'opérateur de concaténation & permet de juxtaposer deux objets de type std_logic ou std_logic_vector.

Exemples :

```
signal octet1, octet2 : std_logic_vector(7 downto 0) ;
signal mot1, mot2 : std_logic_vector(15 downto 0) ;
mot1 <= octet1&octet2 ;
mot2 <= "100"&octet1(3 downto 0)&octet2&'1' ;
```

Il permet de simplifier l'écriture de :

```
if (A = '1' and B = '0') then …
```

en écrivant :

```
if (A&B = "10") then …
```