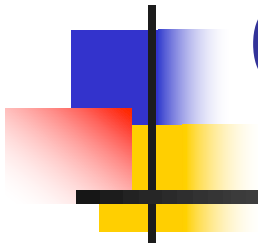


C6 – Les Types

Yann DOUZE

VHDL

Les Types énumérés

- Définition d'un nouveau type de donnée

```
type Opcode is (Add, Neg, Load, Store, Jmp, Halt);

signal S: Opcode;
```

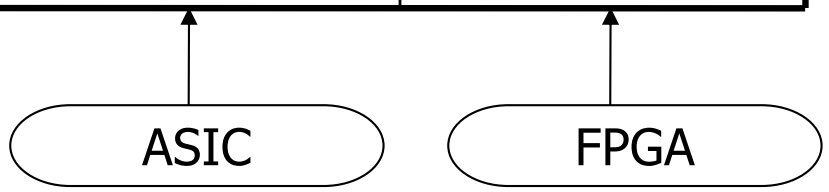
```
S <= Add;
```

```
process(S)
begin
case S is
when Add =>
        ...
```

Synthèse des types énumérés

- Encodage du type énuméré pour la synthèse

	Binaire	One hot
Add	000	100000
Neg	001	010000
Load	010	001000
Store	011	000100
Jmp	100	000010
Halt	101	000001



Les Types définit par défaut

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

Inclut par défaut

```
INTEGER      0  1  99
BIT          '0'  '1'
BIT_VECTOR   "101011"
BOOLEAN      FALSE  TRUE
STRING       "Hello World"
TIME         10 NS
REAL         1.345
```

Types du package STD.STANDARD

Les types logiques à valeurs multiple

Dans le package STD.STANDARD

```
type BOOLEAN is (False, True);

type BIT is ('0', '1');
```

Dans le package IEEE.STD_LOGIC_1164

```
type STD_ULOGIC is (
'U',    -- non initialisé (par défaut)
'X',    -- état inconnu
'0',    -- 0 puissant
'1',    -- 1 puissant
'Z',    -- Haute impédance
'W',    -- État inconnu mais faible
'L',    -- 0 faible
'H',    -- 1 faible
'-');   -- indifférent (pour la synthèse)

subtype STD_LOGIC is RESOLVED STD_ULOGIC;
```

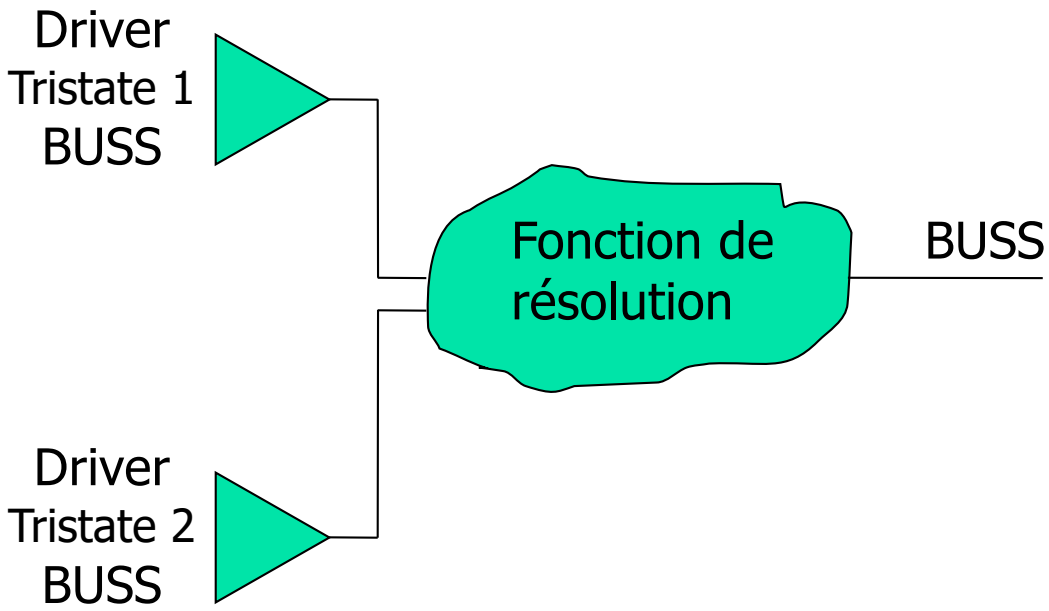
Résolution : définit la priorité

Drivers et Fonction de Résolution

```
subtype STD_LOGIC is RESOLVED STD_ULOGIC;
```

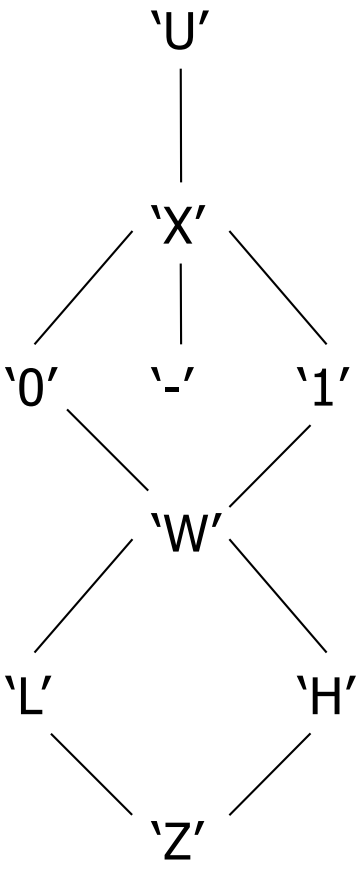
```
Signal BUSS,ENB1,ENB2,D1,D2: STD_LOGIC;
...
TRISTATE1: process (ENB1,D1)
Begin
if ENB1 = '1' then
        BUSS <= D1;
else
        BUSS <= 'Z';
end if;
End process;

TRISTATE2: process (ENB2,D1)
Begin
   if ENB2 = '1' then
      BUSS <= D2;
   else
      BUSS <= 'Z';
   end if;
End process;
```



Résolution de STD_LOGIC

- Le type STD_LOGIC est définit dans le package STD_LOGIC_1164
- Les Valeurs les plus hautes dans le schéma sont prioritaires



Valeurs initiales

- Signaux et variables sont initialisés au début d'une simulation.
- La valeur par défaut est la valeur la plus à gauche du type.
- Attention : La synthèse ignore les valeurs initiales.

```
type Opcode is (Add, Neg, Load, Store, Jmp, Halt);
signal S: Opcode;
signal CLOCK, RESET: STD_LOGIC;
variable V1: STD_LOGIC_VECTOR(0 to 1);
variable V2: STD_LOGIC_VECTOR(0 to 1):="01";
signal N: Opcode:= Halt;
constant size: INTEGER := 16;
constant ZERO: STD_LOGIC_VECTOR := "0000";
```

Valeur initiale Add

Valeur initiale 'U'

Valeur initiale 'UU'

Relation implicite des opérateurs

- Pour le type STD_LOGIC

```
'U' < 'X' < '0' < '1' < 'Z' < 'w' < 'L' < 'H' < '-'
```

- Pour le type STD_LOGIC_VECTOR

```
'0' < "00" < "000" < "001" < "100" < "111" < "1111"
```

- Les operations de comparaison (<, > , =,…)

Opérations arithmétiques

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ADDER is
   port ( A,B : in STD_LOGIC_VECTOR(7 downto 0);
          SUM : out STD_LOGIC_VECTOR(7 downto 0));
end entity;

architecture A1 of ADDER is
begin
   SUM <= A + B;
end architecture;
```

Erreur: "+" n'est pas définit pour le type STD_LOGIC_VECTOR

Utilisation de NUMERIC_STD

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity ADDER is
   port (      A,B : in UNSIGNED(7 downto 0);
               SUM : out UNSIGNED(7 downto 0);
end entity;

architecture A1 of ADDER is
begin
   SUM <= A + B;
end architecture;
```

IEEE Std 1076.3

Opérateur "+" est surchargé pour les types UNSIGNED et SIGNED

Problématique ?

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity ADDER is
   port ( A,B : in STD_LOGIC_VECTOR(7 downto 0);
          SUM : out STD_LOGIC_VECTOR(7 downto 0));
end entity;

architecture A1 of ADDER is
begin


end architecture;
```

Comment faire l'addition de A et B si ils sont de type std_logic_vector ?

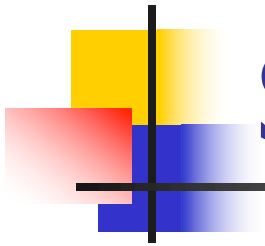
Conversion de type

```
Signal U: UNSIGNED(7 downto 0);
Signal S: SIGNED (7 downto 0);
Signal V: STD_LOGIC_VECTOR(7 downto 0);
```

Conversion entre des types qui sont proches

```
U <= UNSIGNED(S);
S <= SIGNED(U);
U <= UNSIGNED(V);
S <= SIGNED(V);
V <= STD_LOGIC_VECTOR(U);
V <= STD_LOGIC_VECTOR(S);
```

Le nom du type est utilisé pour la conversion de type



Solution

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity ADDER is
   port ( A,B : in STD_LOGIC_VECTOR(7 downto 0);
          SUM : out STD_LOGIC_VECTOR(7 downto 0));
end entity;

architecture A1 of ADDER is
begin
   SUM <= STD_LOGIC_VECTOR(UNSIGNED(A) + UNSIGNED(B));
   -- ou SUM <= STD_LOGIC_VECTOR(SIGNED(A) + SIGNED(B));
end architecture;
```

Fonctions de conversion

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
```

```
Signal U: UNSIGNED(7 downto 0);
Signal S: SIGNED (7 downto 0);
Signal N: INTEGER;
```

Opérations de conversion

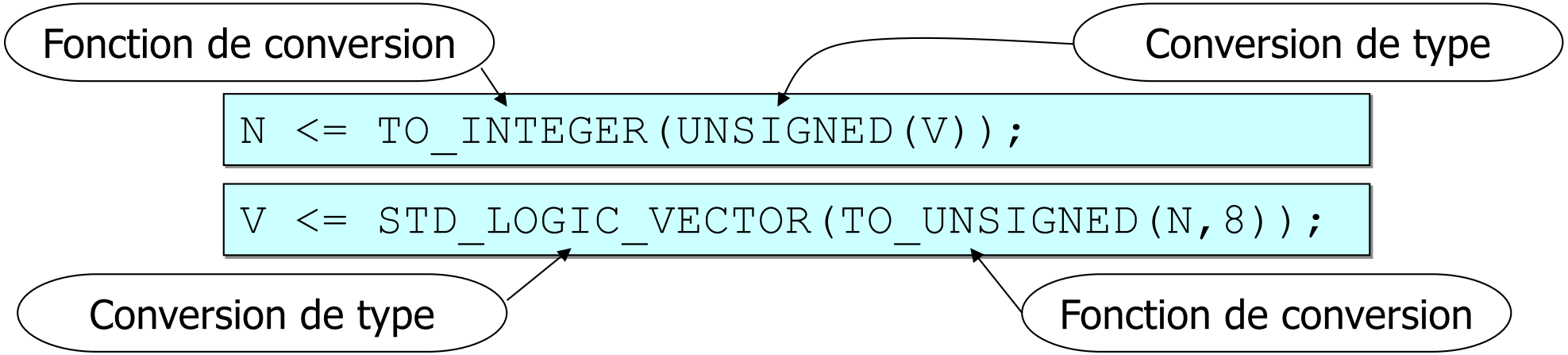
```
N <= TO_INTEGER(U);
N <= TO_INTEGER(S);
U <= TO_UNSIGNED(N,8);
S <= TO_SIGNED(N,8);
```

Taille du vecteur d'arrivée

Conversion entre un INTEGER et un STD_LOGIC_VECTOR

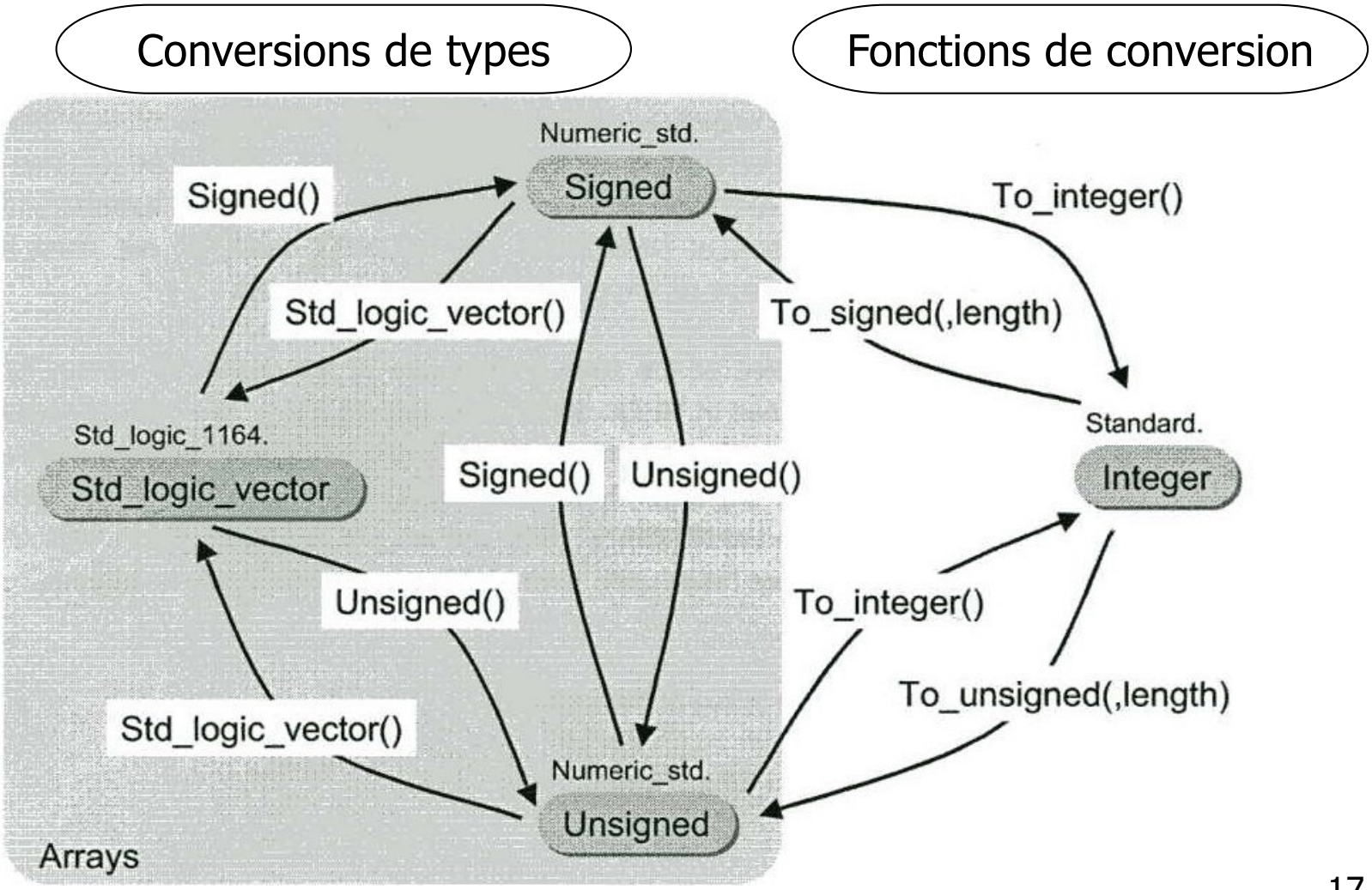
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
```

```
Signal V: STD_LOGIC_VECTOR(7 downto 0);
Signal N: INTEGER;
```




```
V <= STD_LOGIC_VECTOR(SIGNED(V)+1);
```

Tableau Récapitulatif



Sommaire de NUMERIC_STD

```
+   -   *   /   rem   mod
<   <=   >   >=   =  /=

UNSIGNED x UNSIGNED
UNSIGNED x NATURAL
NATURAL x UNSIGNED
SIGNED x SIGNED
SIGNED x INTEGER
INTEGER x SIGNED
```

```
sll   srl   rol   ror

UNSIGNED x UNSIGNED
SIGNED x INTEGER
```

```
not   and   or   nand nor xor
xnor

UNSIGNED x UNSIGNED
SIGNED x INTEGER
```

```
TO_INTEGER  [UNSIGNED                ] return INTEGER
TO_INTEGER  [SIGNED                  ] return INTEGER
TO_UNSIGNED [NATURAL, NATURAL        ] return UNSIGNED
TO_SIGNED   [INTEGER, NATURAL        ] return SIGNED
RESIZE      [UNSIGNED,NATURAL        ] return UNSIGNED
RESIZE      [SIGNED,  NATURAL        ] return SIGNED
```

Signature (VHDL 93)

Exercice 1 (Addition de A,B et C)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

Entity ADDER is
port( A      : in STD_LOGIC_VECTOR(7 downto 0);
      B      : in INTEGER;
      C      : in SIGNED(7 downto 0));
      SUM    : out STD_LOGIC_VECTOR(7 downto 0);
end entity;

Architecture BEHAVIOUR of ADDER is
Begin
SUM <=

End architecture;
```

Exercice 2 (Multiplexeur 8 vers 1)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity Mux8to1 is
port( Address :    in STD_LOGIC_VECTOR(2 downto 0);
      IP      :    in STD_LOGIC_VECTOR(7 downto 0);
      OP      :    out STD_LOGIC);
end entity;
architecture BEHAVIOUR of Mux8to1 is
begin

OP <=IP(                                (Address)));

end architecture ;
```

Package STD_LOGIC_UNSIGNED/SIGNED

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
-- use IEEE.STD_LOGIC_SIGNED.all;
```

Ne pas utiliser en même temps

+ -	*	< <= > >= = /=
STD_LOGIC_VECTOR STD_ULOGIC INTEGER	STD_LOGIC_VECTOR	STD_LOGIC_VECTOR INTEGER

```
CONV_INTEGER[STD_LOGIC_VECTOR] return INTEGER
```

Package STD_LOGIC_ARITH

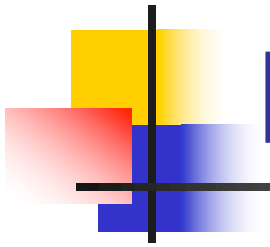
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
```

+ -
STD_ULOGIC
UNSIGNED
SIGNED
INTEGER

*
UNSIGNED
SIGNED

< <= > >= = /=
UNSIGNED
SIGNED
INTEGER

```
CONV_INTEGER[INTEGER/UNSIGNED/SIGNED/STD_ULOGIC] return INTEGER
CONV_UNSIGNED[INTEGER/UNSIGNED/SIGNED/STD_ULOGIC, INTEGER] return UNSIGNED
CONV_SIGNED[INTEGER/UNSIGNED/SIGNED/STD_ULOGIC, INTEGER] return SIGNED
CONV_STD_LOGIC_VECTOR[INTEGER/UNSIGNED/SIGNED/STD_ULOGIC, INTEGER] return
STD_LOGIC_VECTOR
EXT[STD_LOGIC_VECTOR, INTEGER] return STD_LOGIC_VECTOR
SXT[STD_LOGIC_VECTOR, INTEGER] return STD_LOGIC_VECTOR
```



Exercice

- Faire l’exercice du C6