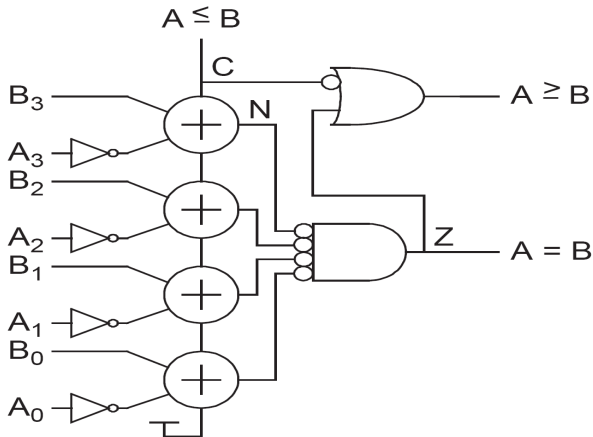


COMPAREUR

FONCTIONALITÉ ET SPÉCIFICATIONS

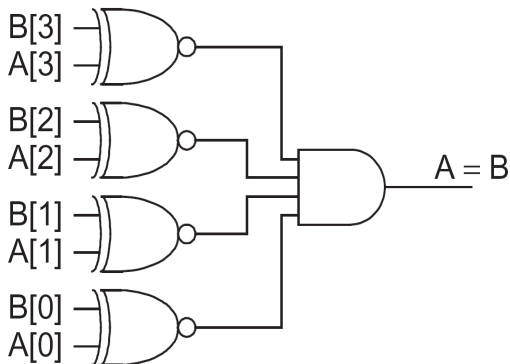
- Utilisation d'un soustracteur modifié



COMPAREUR

FONCTIONALITÉ ET SPÉCIFICATIONS

- Si on veut calculer uniquement si $A=B$, le circuit est plus simple



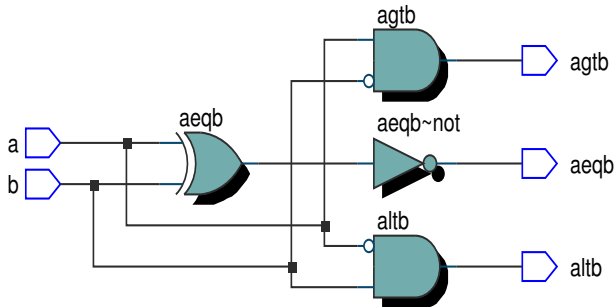
COMPAREUR

DESCRIPTION VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity Compareur is
port(
    a,b           : in std_logic;
    agtb, altb, aeqb : out std_logic);
end Compareur;
architecture archConc of Compareur is
begin
    agtb <= '1' when a > b else '0';
    altb <= '1' when a < b else '0';
    aeqb <= '1' when a = b else '0';
end archConc;
```

COMPAREUR

DESCRIPTION VHDL



COMPAREUR

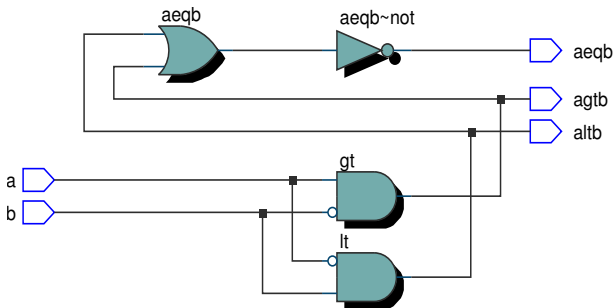
DESCRIPTION VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity Compareur is
port(
    a,b           : in std_logic;
    agtb, altb, aeqb : out std_logic);
end Compareur;
architecture archConc1 of Compareur is
    signal gt, lt :std_logic;
begin
    gt <= '1' when a > b else '0';
    lt <= '1' when a < b else '0';
    agtb <= gt;
    altb <= lt;
```

COMPAREUR

DESCRIPTION VHDL

```
aeqb <= not (gt or lt);  
end archConc1;
```



COMPAREUR

DESCRIPTION VHDL

- Quelles sont les différences au niveau RTL entre ces deux descriptions VHDL ?
- A votre avis, laquelle des deux est la plus rapide ?

SOMMAIRE

TYPES DE DONNÉES

- Définition d'un type de données
 - ▷ un ensemble de valeurs pouvant être affectées à un objet
 - ▷ un ensemble d'opérations pouvant être appliquées sur les objets d'un même type
- VHDL est un langage très typé
- un objet peut être affecté uniquement avec la valeur du même type
- uniquement les opérations définies pour un type de données peuvent être appliquées sur un objet de même type

Types de données standard :

- entier (**integer**) :
 - ▷ de $-(2^{31} - 1)$ à $2^{31} - 1$ par défaut (au maximum)
 - ▷ deux sous-types : **natural** et **positive**

TYPES DE DONNÉES

- ▷ possibilité de spécifier un domaine avec **range** :
`variable UnEntier : integer range 0 to 511;`
- ▷ possibilité d'accéder aux limites du domaine avec des attributs : `signal limB, limH : natural;`
`begin limB <= UnEntier'Left; limH <=`
`UnEntier'Right;`
⇒ limB vaut 0 et limH vaut 511
- booléen (boolean) : (false, true)
- bit : ('0', '1')
- bit_vector : un tableau 1D de bits

TYPES DE DONNÉES

operator	description	data type of operand a	data type of operand b	data type of result
a ** b	exponentiation	integer	integer	integer
abs a	absolute value	integer		integer
not a	negation	boolean, bit, bit_vector		boolean, bit, bit_vector
a * b	multiplication	integer	integer	integer
a / b	division			
a mod b	modulo			
a rem b	remainder			
+ a	identity	integer		integer
- a	negation			
a + b	addition	integer	integer	integer
a - b	subtraction			
a & b	concatenation	1-D array, element	1-D array, element	1-D array

TYPES DE DONNÉES

a sll b	shift left logical	bit_vector	integer	bit_vector
a srl b	shift right logical			
a sla b	shift left arithmetic			
a srl b	shift right arithmetic			
a rol b	rotate left			
a ror b	rotate right			
a = b	equal to	any	same as a	boolean
a /= b	not equal to			
a < b	less than	scalar or 1-D array	same as a	boolean
a <= b	less than or equal to			
a > b	greater than			
a >= b	greater than or equal to			
a and b	and	boolean, bit,	same as a	boolean, bit,
a or b	or	bit_vector		bit_vector
a xor b	xor			
a nand b	nand			
a nor b	nor			
a xnor b	xnor			

TYPES DE DONNÉES

`std_logic`

- Pourquoi le type `bit` n'est pas suffisant ?
- le package `std_logic_1164`
- `std_logic` :
- 9 valeurs possibles :
 - ▷ '0' ou '1'
 - ▷ 'Z' : l'état de haute impédance
 - ▷ 'L' ou 'H' : un faible '0' ou '1'
 - ▷ 'X', 'W' : inconnu ou un faible inconnu
 - ▷ 'U' : non initialisé
 - ▷ '-' : peu importe (indéfini)
- `std_logic_vector` : vecteur de `std_logic`
→ `std_logic_vector(7 downto 0)`

TYPES DE DONNÉES

std_logic

- utilisation :

```
library ieee;
use ieee.std_logic_1164.all;
```


Opérateurs utilisés avec le type std_logic

overloaded operator	data type of operand a	data type of operand b	data type of result
not a	std_logic_vector std_logic		same as a
a and b			
a or b			
a xor b	std_logic_vector	same as a	same as a
a nand b	std_logic		
a nor b			
a xnor b			

TYPES DE DONNÉES

std_logic

Les fonctions de conversion disponibles dans le package :

function 	data type of operand a	data type of result
to_bit(a)	std_logic	bit
to_stdulogic(a)	bit	std_logic
to_bit_vector(a)	std_logic_vector	bit_vector
to_stdlogicvector(a)	bit_vector	std_logic_vector

Exemple :

TYPES DE DONNÉES

std_logic

```
signal s1, s2, s3: std_logic_vector(7 downto 0);  
signal b1, b2      : bit_vector(7 downto 0);
```

--KO

```
s1 <= b1;  
b2 <= s1 and s2;  
s3 <= b1 or s2;
```

--OK

```
s1 <= to_stdlogicvector(b1);  
b2 <= to_bitvector(s1 and s2);  
s3 <= to_stdlogicvector(b1) or s2;  
-- ou  
s3 <= to_stdlogicvector(b1 or to_bitvector(s2));
```


OPÉRATEURS SUR LES TABLEAUX D'ÉLÉMENTS

- Les opérandes n'ont pas toujours la même taille
- Lors de la comparaison de deux tableaux n'ayant pas la même taille, tous les éléments sont comparés un par un en partant de gauche (LSB)
- Exemple :

"011"="011", "011">"010", "011">"00010", "0110">"011"

Tous les exemples précédents sont vrais

- Opérateur de concaténation

```
y<= "00" & a(7 downto 2);
```

```
y<= a(7) & a(7) & a(7 downto 2);
```

```
y<= a(1 downto 0) & a(7 downto 2);
```

OPÉRATEURS SUR LES TABLEAUX D'ÉLÉMENTS

□ Opération d'agrégation

```
a <= "10100000";  
a <= (7=> '1', 6=> '0', 0=> '0', 1=> '0', 5=> '1',  
      4=> '0', 3=> '0', 2=> '1');  
a <= (7|5=> '1', 6|4|3|2|1|0 => '0');  
a <= (7|5=> '1', others => '0');  
a <= "00000000";  
a <= (others => '0');
```

TYPES DE DONNÉES

PACKAGE `numeric_std`

- Comment réaliser les opérations arithmétiques avec les `std_logic` ?
- la solution : le package `numeric_std`
- définit un entier comme un tableau d'éléments de type `std_logic`
- Deux types : `unsigned` et `signed`
- utilisation :

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

Les opérateurs définis dans le package :

TYPES DE DONNÉES

PACKAGE numeric_std

overloaded operator	description	data type of operand a	data type of operand b	data type of result
abs a - a	absolute value negation	signed		signed
a * b a / b a mod b a rem b a + b a - b	arithmetic operation	unsigned unsigned, natural signed signed, integer	unsigned, natural unsigned signed, integer signed	unsigned unsigned signed signed
a = b a /= b a < b a <= b a > b a >= b	relational operation	unsigned unsigned, natural signed signed, integer	unsigned, natural unsigned signed, integer signed	boolean boolean boolean boolean

TYPES DE DONNÉES

PACKAGE numeric_std

```
signal a,b,c,d: unsigned (7 downto 0);  
...  
a <= b + c;  
d <= b + 1;  
e <= (5 + a + b) - c;
```

TYPES DE DONNÉES

PACKAGE numeric_std

function	description	data type of operand a	data type of operand b	data type of result
shift_left(a,b)	shift left	unsigned, signed	natural	same as a
shift_right(a,b)	shift right			
rotate_left(a,b)	rotate left			
rotate_right(a,b)	rotate right			
resize(a,b)	resize array	unsigned, signed	natural	same as a
std_match(a,b)	compare '-'	unsigned, signed std_logic_vector, std_logic	same as a	boolean
to_integer(a)	data type	unsigned, signed		integer
to_unsigned(a,b)	conversion	natural	natural	unsigned
to_signed(a,b)		integer	natural	signed

TYPES DE DONNÉES

PACKAGE `numeric_std`

- Les types de données `std_logic_vector`, `unsigned` ou `signed` sont définis comme des tableaux d'éléments `std_logic`
- Ces trois types sont considérés comme des types différents
- Utilisation de fonctions de conversion pour passer d'un type à un autre

TYPES DE DONNÉES

PACKAGE numeric_std

data type of a	to data type	conversion function / type casting
unsigned, signed	std_logic_vector	std_logic_vector(a)
unsigned, std_logic_vector	unsigned	unsigned(a)
unsigned, signed	std_logic_vector	std_logic_vector(a)
unsigned, signed	integer	to_integer(a)
natural	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)

TYPES DE DONNÉES

PACKAGE numeric_std

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
. . .
signal s1, s2, s3, s4, s5, s6: std_logic_vector(3
    downto 0);
signal u1, u2, u3, u4, u6, u7: unsigned(3 downto 0);
signal sg: signed(3 downto 0);

-- OK
u3 <= u2 + u1;    --- ok, operandes non-signés
u4 <= u2 + 1;     --- ok, operandes non-signé et natural
```

TYPES DE DONNÉES

PACKAGE numeric_std

--KO

```
u5 <= sg;    -- type mismatch
u6 <= 5;     -- type mismatch
```

--Solution

```
u5 <= unsigned(sg);    -- type casting
u6 <= to_unsigned(5,4); -- fonction de conversion
```

--KO

```
s3 <= u3;    -- type mismatch
s4 <= 5;     -- type mismatch
```

--Solution

```
s3 <= std_logic_vector(u3); -- type casting
```

TYPES DE DONNÉES

PACKAGE numeric_std

```
s4 <= std_logic_vector(to_unsigned(5,4));
```

--KO

```
s5 <= s2 + s1; + indefini pour std_logic_vector
```

```
s6 <= s2 + 1; + indefini
```

-- Solution

```
s5 <= std_logic_vector(unsigned(s2) + unsigned(s1));
```

```
s6 <= std_logic_vector(unsigned(s2) + 1);
```

TYPES DE DONNÉES

PACKAGE `std_logic_arith`

- package développé par *Synopsys* avant le standard IEEE `numeric_std`
- presque similaire à `numeric_std`
- deux nouveaux types : `unsigned` et `signed`
- les détails d'implémentation sont différents
- manipule les `std_logic_vector` comme des nombres signés ou non-signés
- Dans les outils de simulation, on le trouve souvent dans la library `ieee` (même s'il n'en fait pas partie)

TYPES DE DONNÉES

PACKAGE std_logic_arith

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_arith_unsigned.all;  
...  
signal s1,s2,s3,s4,s5,s6: std_logic_vector(3 downto  
    0);  
...  
s5<=s2+s1; -- ok, l'opérateur + surchargé  
s6<= s2+1; -- ok, l'opérateur + surchargé
```

- un seul des deux packages peut être utilisé à la fois
- leur utilisation remet en cause la réputation du langage VHDL comme un langage très typé

TYPES DE DONNÉES

PACKAGE `std_logic_arith`

- En conclusion : préférer le package `numeric_std`

EXERCICES

UTILISATION DU PACKAGE `numeric_std`

- ➊ Réaliser un additionneur 4 bits en instanciant l'additionneur 1 bit déjà présenté
- ➋ Décrire un additionneur 4 bits en utilisant le package `numeric_std`
- ➌ Décrire un soustracteur 4 bits en utilisant le package `numeric_std`
- ➍ Décrire un comparateur 8 bits en utilisant le package `numeric_std`
- ➎ Pour le comparateur 8 bits, est-il nécessaire de faire des conversions en `unsigned` ?

EXERCICES DE SYNTHÈSE

CODAGE D'UNE ALU SIMPLE

- 1 Donnez un code permettant de synthétiser une ALU opérant sur des données de 8 bits et possédant les 5 modes suivants : ADD; SUB; AND; OR; XOR
- 2 Modifiez le code pour que la taille des données soit générique
- 3 Observez le résultat de synthèse dans la vue RTL
- 4 Écrivez et simulez cette ALU avec un testbench permettant de vérifier les 5 modes sur 4 vecteurs d'entrée chacun