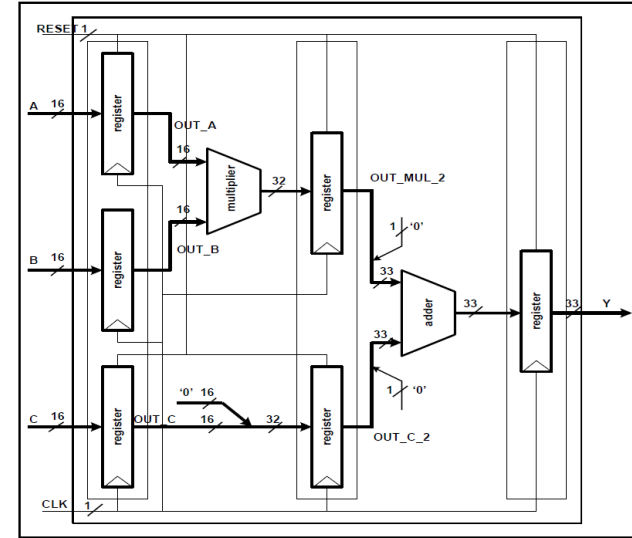
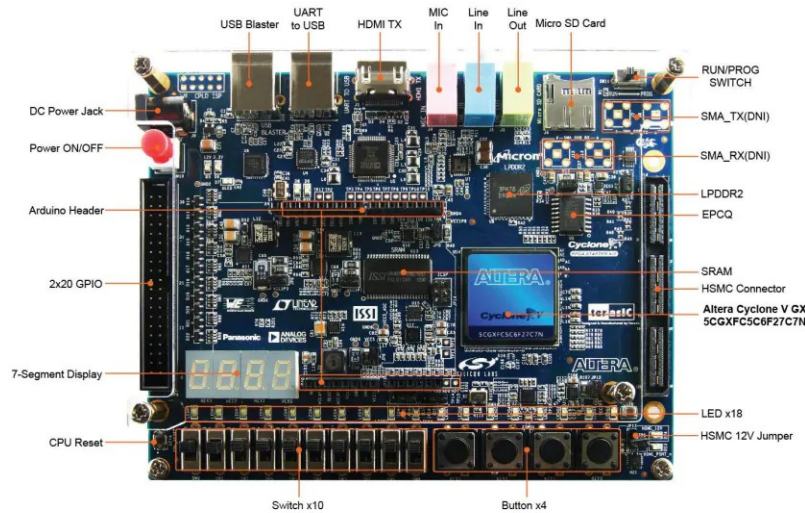


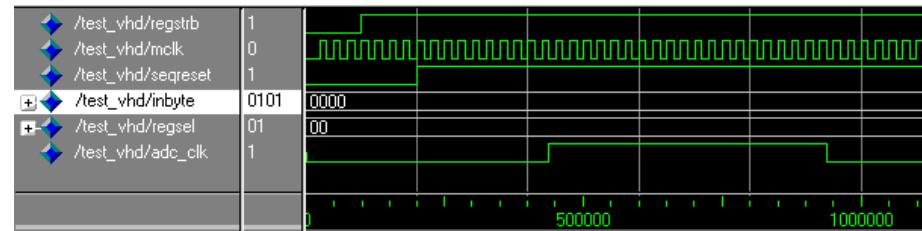
UE 308 TP : Séance 2 : Description d'un circuit numérique, langage VHDL, FPGA



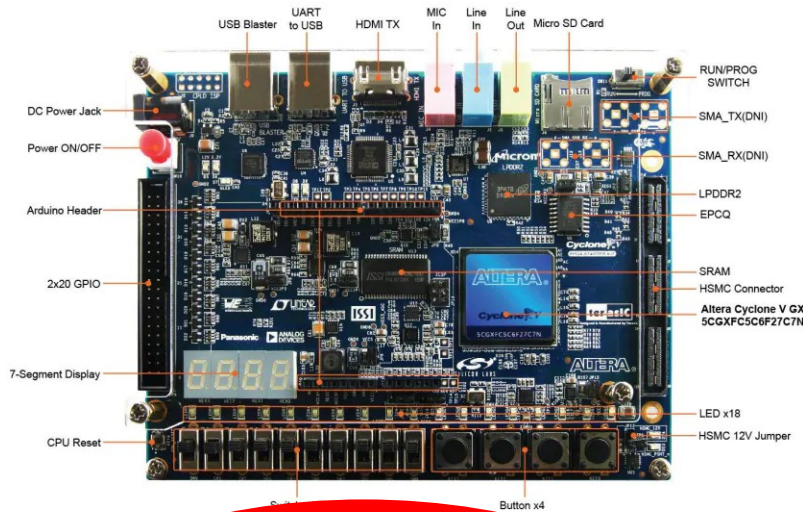
```

31 ClkDivP : process(Mclk,SeqReset)
32 begin
33   if SeqReset = '0' then
34     ADCClk <= '0';
35     ADC_div <= "001001";
36   elsif Mclk = '0' and Mclk'event then
37     if ADC_div = "000000" then
38       ADCClk <= not(ADCClk);
39       case ClkSel is
40         when "000" => -- 20MHz - divide by 2
41           when "001" => -- 10MHz
42             ADC_div <= "000001"; -- divide by 4
43           when "010" => -- 4MHz
44             ADC_div <= "000100"; -- divide by 10
45           when "011" => -- 2MHz
46             ADC_div <= "001001"; -- divide by 20
47           when "100" => -- 1MHz
48             ADC_div <= "001001"; -- divide by 40
49           when others => -- 400KHz
50             ADC_div <= "010011"; -- divide by 100
51         end case;
52       else
53         ADC_div <= (unsigned(ADC_div) - 1);
54       end if;
55     end if;
56   end process; -- ClkDivP

```



Language VHDL : Description hardware (différent de la programmation software)



Le VHDL est un langage à part, plus proche de l'électronique que de l'informatique. Il n'est d'ailleurs pas rare de voir implémenté sur des FPGA des architectures de micro-contrôleurs, eux-mêmes programmés en assembleur ou en C dans la suite d'un projet.

L'environnement Quartus permet également de faire de la simulation/test (programmation software).

```
31  divP : process(Mclk,SeqReset)
32  begin
33  if SeqReset = '0' then
34  ADCClk <= '0';
35  ADC_div <= "001001";
36  elsif Mclk = '0' and Mclk'event then
37  if ADC_div = "000000" then
38  ADCClk <= not(ADCClk);
39  case ClkSel is
40  when "000" => -- 20MHz - divide by 2
41  when "001" => -- 10MHz
42  ADC_div <= "000001"; -- divide by 4
43  when "010" => -- 4MHz
44  ADC_div <= "000100"; -- divide by 10
45  when "011" => -- 2MHz
46  ADC_div <= "001001"; -- divide by 20
47  when "100" => -- 1MHz
48  ADC_div <= "001001"; -- divide by 40
49  when others => -- 400KHz
50  ADC_div <= "010011"; -- divide by 100
51  end case;
52  else
53  ADC_div <= (unsigned(ADC_div) - 1);
54  end if;
55  end if;
56  end ;
```

TP1 et TP2 : circuits combinatoires

Semaine prochaine TP3 : circuits séquentiels

Language VHDL

Exemple de structure d'une description

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.numeric_std.all;
```

```
ENTITY porte is  
PORT(  
    a,b : IN STD_LOGIC;  
    s : OUT STD_LOGIC  
);  
END porte;
```

```
ARCHITECTURE archi OF porte IS  
BEGIN  
    s<=a AND b;  
END archi;
```

Language VHDL

Exemple de structure d'une description

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.numeric_std.all;
```

```
ENTITY porte is  
  PORT(  
    a,b : IN STD_LOGIC;  
    s : OUT STD_LOGIC  
  );  
END porte;
```

```
ARCHITECTURE archi OF porte IS  
  BEGIN  
    s<=a AND b;  
  END archi;
```



Définition des différents signaux (par exemple STD_LOGIC et STD_LOGIC_VECTOR) et des opérateurs associés (and, or etc.)

Définition des différents types (entiers, signés etc.) et de leurs conversions, des opérations arithmétiques de base (+,-), de la comparaison (>,<) etc.

Language VHDL

Exemple de structure d'une description

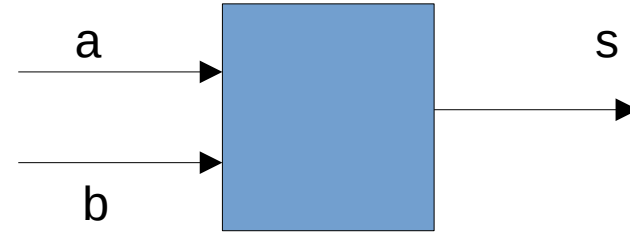
```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.numeric_std.all;
```

```
ENTITY porte is  
  PORT(  
    a,b : IN STD_LOGIC;  
    s : OUT STD_LOGIC  
  );  
END porte;
```

```
ARCHITECTURE archi OF porte IS  
BEGIN  
  s<=a AND b;  
END archi;
```



Description **externe** du circuit



STD_LOGIC et STD_LOGIC_VECTOR sont des types à **9 états** (0 fort, 1 fort, 0 faible, 1 faible, haute impédance, **inconnu**, non initialisé etc.). 3 états physiques + 6 états pour la simulation.

STD_LOGIC_VECTOR est un bus (de bits) (le poids fort est toujours à gauche)

Eventuellement on peut trouver dans l'« entity » des paramètres « génériques », par exemple

```
entity Example is  
  generic (  
    data_width : integer := 8;  
  );  
  port (  
    e : in std_logic_vector(data_width-1 downto 0);  
    ...  
  );  
end Example;
```

Language VHDL

Exemple de structure d'une description

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.numeric_std.all;
```

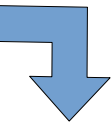
```
ENTITY porte is  
PORT(  
    a,b : IN STD_LOGIC;  
    s : OUT STD_LOGIC  
);  
END porte;
```

```
ARCHITECTURE archi OF porte IS  
BEGIN  
    s<=a AND b;  
END archi;
```



Description **interne** du circuit

Il en existe plusieurs pour un même circuit



Language VHDL

Exemple de structure d'une description

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.numeric_std.all;
```

```
ENTITY porte is  
PORT(  
    a,b : IN STD_LOGIC;  
    s : OUT STD_LOGIC  
);  
END porte;
```

```
ARCHITECTURE archi OF porte IS  
BEGIN  
    s<=a AND b;  
END archi;
```

OU

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;
```

```
ENTITY porte is  
PORT(  
    e : IN STD_LOGIC_VECTOR(1 downto 0);  
    s : OUT STD_LOGIC);  
END porte;
```

```
ARCHITECTURE archi OF porte IS  
BEGIN  
    PROCESS(e)  
    BEGIN  
        CASE(e) is  
            WHEN "00" => s <= '0';  
            WHEN "01" => s <= '0';  
            WHEN "10" => s <= '0';  
            WHEN "11" => s <= '1';  
            WHEN OTHERS => s <= '0';  
        END CASE;  
    END PROCESS;  
END archi;
```

Language VHDL

Architecture —> Partie déclarative optionnelle (déclaration de variables et signaux « internes »)

Description comportementale

—> Instructions **concurrentes** (with select, etc.)

—> Processus (ou process) : Instructions **séquentielles**

Case, when
If, else
For



Dans une architecture, des instructions concurrentes s'exécutent en permanence et en même temps (description hardware). Attention aux courts circuits !

Ce n'est pas parce que les instructions sont séquentielles dans un processus que nous sommes en train de faire de la logique séquentielle (voir exemple porte ET ci-avant et exemple ci-après).

Bien que le process contienne des instructions séquentielles, il est lui même une instruction concurrente s'exécutant en parallèle avec les autres process ou instructions concurrentes (voir exemple ci-après).

Language VHDL

Structure de process

Un process peut être dans deux états : « endormi » ou « réveillé »

Pour passer de l'état « endormi » à « réveillé » on spécifie **une liste de sensibilité** (vide si on utilise une instruction wait) composé des signaux. On passe de l'état « réveillé » à « endormi » lorsque le process a terminé d'exécuter sa dernière instruction.

Les process seront indispensables pour décrire les circuits séquentiels synchrone (le front montant du signal d'horloge sera placé dans la liste de sensibilité). **Mais un process peut aussi décrire un circuit combinatoire.**

Porte ET 3ème version + [modif](#)

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

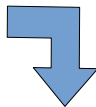
entity porte_et_vhdl is
    port(
        a: in std_logic;
        b: in std_logic;
        c : out std_logic ;
        s: out std_logic
    );
end entity;

architecture dataflow of porte_et_vhdl is
begin
    process(a,b) is
    begin
        if a='1' AND b='1' then
            s <= '1';
        else
            s <= '0';
        end if;
    end process;
    c <= not(a) ; - - modif
end architecture;
```



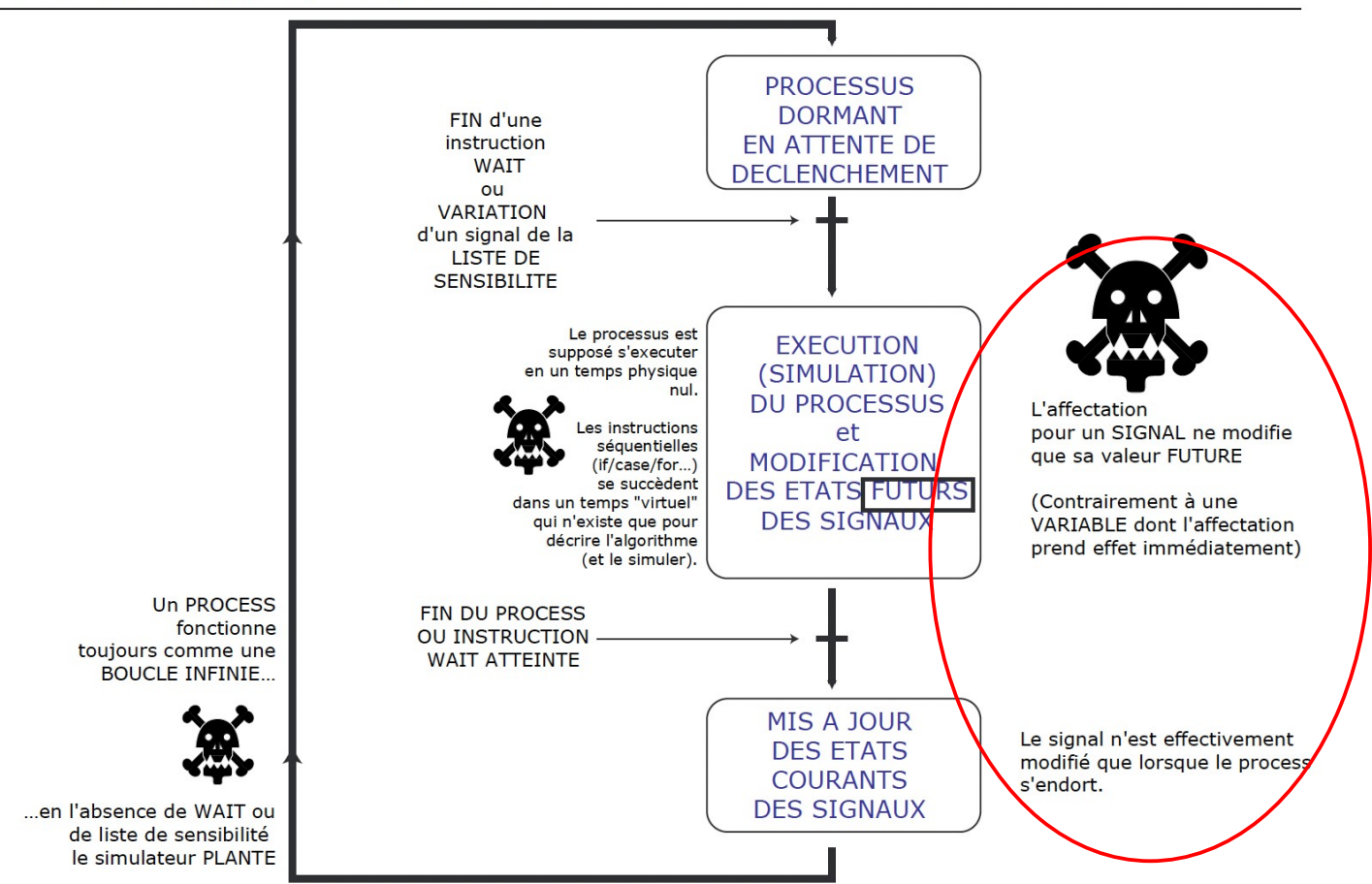
Pour du combinatoire, un process doit contenir **tout les cas possibles** (utilisation de else ou when others). **Sans quoi l'outil de synthèse, considérant que les sorties non assignées conservent leur ancienne valeur, placera des bascules D en sortie de chaque sortie non affectée.** Cette solution est alors très mauvaise, puisqu'elle transforme la fonction en une fonction de logique synchrone

Le temps dans un process est virtuel.



Language VHDL

LE TEMPS DANS LES PROCESSUS



Exemple :

Dans un process si on écrit :

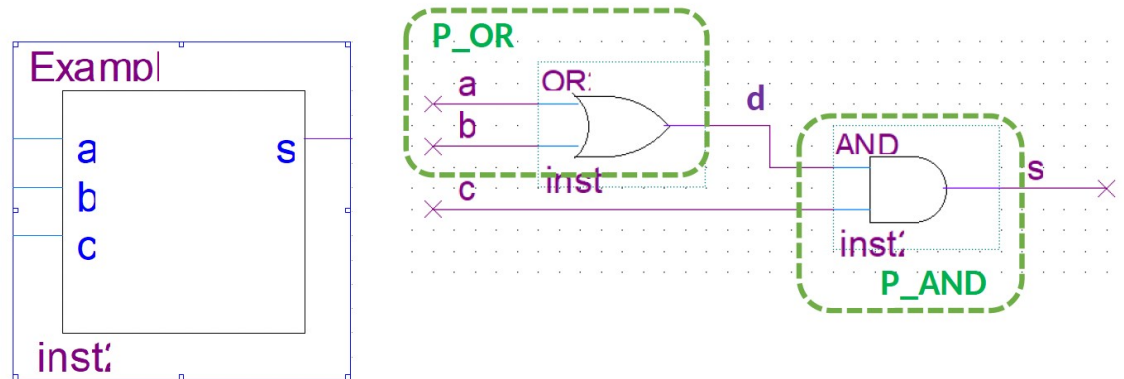
```
A<=A+1 ;  
A<=A+3 ;  
A<=A+2 ;
```

alors A n'est pas augmenté de 6 mais seulement de 2.

Language VHDL

Signal : grandeur physique **qui existe réellement** dans le circuit après compilation (différent des variables). Evidement, les grandeurs présentent dans l'ENTITY sont des signaux (externe). Ici d est un signal interne (permet de structurer le code)


```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY Example IS
6  PORT (
7      a : IN STD_LOGIC;
8      b : IN STD_LOGIC;
9      c : IN STD_LOGIC;
10     s : OUT STD_LOGIC
11 );
12 END Example;
```



```
12  ARCHITECTURE archi2 OF Example IS
13
14     SIGNAL d : STD_LOGIC;
15
16
17  BEGIN
18
19     P_AND: PROCESS(c,d) IS
20     BEGIN
21         IF c='1' AND d='1' THEN
22             s <= '1';
23         ELSE
24             s <= '0';
25         END IF;
26     END PROCESS P_AND;
27
28     P_OR: PROCESS(a,b) IS
29     BEGIN
30         IF a='1' OR b='1' THEN
31             d <= '1';
32         ELSE
33             d <= '0';
34         END IF;
35     END PROCESS P_OR;
36  END archi2;
```

Language VHDL

Types de signaux/variables

Boolean  (Vrai,Faux) format de sortie des opérateurs relationnels (>,<=,/=, etc.) dans les structures conditionnelles (if, when, etc.)

STD_LOGIC (et STD_LOGIC_VECTOR)
Signed et Unsigned (64 bits)

Integer (32 bits)
Natural (31 bits)

Real
Char

Language VHDL

Types de signaux/variables

Base 2

Boolean (Vrai,Faux)

STD_LOGIC (et STD_LOGIC_VECTOR)
Signed et Unsigned (64 bits)



Non « structuré » (bibliothèque ieee.std_logic_1164.all)
Structuré (bibliothèque ieee.numeric_std.all)

Integer (32 bits)
Natural (31 bits)

Les quatre types sont similaires **mais on ne peut pas faire d'arithmétique** (+, - ,etc.) avec les STD_LOGIC et les STD_LOGIC_VECTOR. Il faut convertir en SIGNED ou UNSIGNED.

Real
Char

stackoverflow

AboutProductsFor Teams

Search...

Home

Questions

Tags

Users

Companies

LABS

Discussions

COLLECTIVES

TEAMS

Ask questions, find answers and collaborate at work with Stack Overflow for Teams.

Explore Teams

Create a free Team

Asked 13 years, 5 months agoModified 1 year, 7 months agoViewed 79k times

22

I wanna have a simple module that adds two std_logic_vectors. However, when using the code below with the + operator it does not synthesize.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity add_module is
    port(
        pr_in1 : in std_logic_vector(31 downto 0);
        pr_in2 : in std_logic_vector(31 downto 0);
        pr_out  : out std_logic_vector(31 downto 0)
    );
end add_module;

architecture Behavior of add_module is

begin

    pr_out <= pr_in1 + pr_in2;

end architecture Behavior;
```

The error message I get from XST

Line 17. + can not have such operands in this context.

Do I miss a library? If possible, I do not wanna convert the inputs into natural numbers.

Many thanks

Language VHDL

Types de signaux/variables

Boolean (Vrai,Faux)

Base 10

STD_LOGIC (et STD_LOGIC_VECTOR)
Signed et Unsigned (64 bits)

Arithmétique autorisée

Integer (32 bits)  Entiers relatifs (potentiellement négatif)
Natural (31 bits)  Entiers naturels (positifs ou nul)

librairie [ieee.numeric_std.all](#)

Real
Char

Signaux internes à l'architecture

```
architecture Behavioral of exemplesSyntaxe is
    signal A : STD_LOGIC;
    signal B, C, D, E, F, G : STD_LOGIC;
    signal retenue : STD_LOGIC := '1';
    signal compteurH : STD_LOGIC_VECTOR(9 downto 0);
    signal compteurV : STD_LOGIC_VECTOR(9 downto 0) := (others => '0');
    signal operande1, operande2 : STD_LOGIC_VECTOR(3 downto 0) := "0000";
    signal afficheur7Segments : STD_LOGIC_VECTOR(1 to 7);
    signal X, Y : INTEGER range 0 to 1023;
begin
```

Language VHDL

Types de signaux/variables

Boolean (Vrai,Faux)

STD_LOGIC (et STD_LOGIC_VECTOR)
Signed et Unsigned (64 bits)

Integer (32 bits)
Natural (31 bits)

Real
Char



Norme IEEE 754 (voir cours/TD)

Pour la simulation seulement (ne peut pas être un signal)

Etc.

Language VHDL

Opérateurs logiques (Boolean, SIGNED, UNSIGNED, STD_LOGIC(_VECTOR))

Rq : donc défini dans les deux librairies

Opérateurs relationnels (Tous types **mais en scalaire** pour certains)

Opérateur de concaténation (j'ai oublié, donc je regarde la librairie)

Opérateurs arithmétiques (SIGNED, UNSIGNED, INTEGER)

Attention aux librairies « alternatives »

Opérateur	Description	Résultat
not	Complément Logique Unaire	même type
and	ET logique	même type
or	OU logique	même type
nand	Non-ET	même type
nor	Non-OU	même type
xor	OU exclusif	même type
nxor	NON-OU exclusif (93)	même type

Opérateur	Description	Résultat
=	Égalité	Booléen
≠	Inégalité (différent)	Booléen
<	Inférieur	Booléen
<=	Inférieur ou Égal	Booléen
>	Supérieur	Booléen
>=	Supérieur ou Égal	Booléen

Opérateur	Description	Résultat
&	Concaténation	même type, plus large

Ex "01" & "00" = "0100"

Opérateur	Description	Résultat
+	Addition	dépend
-	Soustraction	dépend
abs	Valeur absolue	Même type
*, **	Multiplication, puissance	dépend
/	Division	dépend
mod	Modulo, sur entiers	dépend
rem	Reste, sur entiers	dépend

Conversions de types

Ici, encore une fois, la librairie « ieee.numeric_std.all » doit être utilisée.

Types importants (cette année)

STD_LOGIC_VECTOR (et STD_LOGIC)

UNSIGNED

SIGNED

INTEGER

12 conversions possibles  voir eCampus