# LA PROGRAMMATION MODULAIRE

## I. LES FONCTIONS : function

### I.1. DEFINITION

VHDL permet de rassembler un ensemble d'instructions sous forme de sous programmes appelés fonctions (function). Elles peuvent être appelées de différents endroits, et elles sont utilisées pour améliorer la lisibilité et exploiter la possibilité de réutilisation du code VHDL.

☞ Une function peut être déclarée dans:

- la partie déclarative d'une architecture.
- une function,
- un package ,
- une procedure ,
- une entity,

☞ Ses paramètres peuvent être exclusivement:

- Un signal;
- Une constant;
- Un File (Fichiers);

Le type par défaut est fixé à constant.

Une function peut avoir plusieurs entrées mais n'a qu'une seule sortie. Ses paramètres sont toujours en mode in, les modes out et inout sont interdits. Elle doit comporter au moins un paramètre de type in et au moins une instruction return. On peut déclarer des variables locales à l'intérieur d'une function. Ces variables perdent leur valeur à chaque sortie de la function et sont initialisées à chaque appel;

☞ VHDL autorise la surcharge des fonctions : Plusieurs fonctions peuvent avoir le même identificateur cependant elles doivent se différenciées par au moins un paramètre formel.

☞ Une fonction n'autorise qu'un traitement combinatoire i.e. qu'elle effectue des calculs en se servant des valeurs actuelles des arguments (entrées) et retourne une seule valeur explicitement à travers l'instruction return. Elle n'autorise pas

- Un contrôle d'evènement (pas de process),
- Le timing (pas d'instruction wait): Cela signifie que les fonctions consomment toujours zéro temps de simulation.
- Elle n'autorise pas non plus l'assignation d'un signal.

☞ Chaque ligne de la fonction est exécutée d'une manière séquentielle

### I.2. COMPOSANTES D'UNE FONCTION

Une function peut être composée, si elle est déclarée dans un package, de deux parties :

1. Une partie déclaration, appellé également prototype, qui ne comprend que le nom de la fonction et les noms et les types de ses paramètres formels ainsi que le type de retour selon la syntaxe suivante :

   function Function_Identifier {(List of formal parameters)} return return_type;

   **Exemples**: Prototypes des fonctions

   function func_1(a,b,x: real) return real;

   function add_Signals(signal In1,In2: real) return real;

   La première function ci dessus est appellée func_1, elle a trois parameters a, b et x, tous de type real. La classe de ces paramètres n'est pas spécifiée explicitement, elle est prise par défaut comme signal. Elle retourne une valeur de type real.

   La deuxième fonction au nom de add_Signals utilise deux paramètres d'entrées. La classe de ces paramètres est explicitement spécifiée par utilisation du mot clé signal.

2. Une partie corps, qui comprend le détail de ce que fait la fonction selon la syntaxe suivante :

```
function Function_Identifier [(List of formal parameters)]  return  return_type   is
    Zone de déclaration
begin
    zone d'instructions séquentielles
    return value;                                    -- value must be of return_type of the function
end nom_fonction;
```

où :

Function_Identifier : désigne le nom de la fonction

List of formal parameters : spécifie les paramètres d'entrée de la fonction. Ils ont la forme :

Parametre_class Parametre_Identifier : Parametre_mode Parametre_type

où :

- Parametre_class peut être :
  - Un signal ⇒ signal signal_name : signal_mode
  - Une constante ⇒ constant constant_name : constant_mode.
  - Un Fichier ⇒ file file_name (pas de mode pour le fichier)

  Si le type file est utilisé, il est nécessaire de spécifier le type des données contenues dans le fichier à ouvrir.
- Parametre_Identifier: Le nom du paramètre

- Parametre_mode peut être :
  - Uniquement in. Les modes out et inout sont interdits : Puisque l'unique mode est input, il est pris par défaut. Il n'est donc pas nécessaire de le préciser explicitement.
  - Les fichiers n'ont pas de mode
- Parametre_type peut être std_logic, integer,....

Tous les paramètres formels sont locaux à la fonction et le prototype de celle ci n'existe pas si la fonction est déclarée dans le corps d'une architecture ou d'une entity.

Exemples : Body of functions

```
function Max_Values ( a, b : in real) return real is
begin
    if a<b then
        return b ;
    else
        return a ;
    end if;
end max_values;
```

### I.3. APPEL ET PASSAGE DES PARAMETRES A UNE FUNCTION

L'appel d'une function se fait à travers une expression, séquentielle ou concurrente, n'importe où dans l'architecture qui en possède l'accés. Parce que la fonction retourne un objet, elle devrait figurée dans le coté droit de l'opérateur d'affectation <=. Le retour s'effectue d'une manière explicite par l'exécution de l'instruction return (expression),

### I.4. UTILISATION DES FONCTIONS

Une function (comme une procedure) peut être utilisée selon 3 méthodes (comme pour les component)

1. Décrite dans la zone déclarative de l'architecture ou l'entity qui l'utilise: seul le corps de la fonction est donné (il n y a pas de déclaration)
2. Déclarée et décrite dans un package : La déclaration de la function est faite dans la la partie déclarative du package et le corps de la function est donné dans le corps (body) du package. On inclut le package, dans le projet, pour pouvoir appeler la fonction est le tour est joué;
3. Décrite dans un fichier (dans une Library quelconque) : Ici il faut déclarée la function dans de la zone déclarative de l'architecture qui l'utilise. Il faut bien sûr inclure la bibliothèque qui contient cette function dans le projet pour pouvoir l'appeller.

☞ Peuvent être utilisés dans le monde concurrent ou séquentiel.

Ci-dessous ne sera décrit que la première technique.

## I.5. UTILISATION DES FONCTIONS SANS DECLARATION (SANS PROTOTYPE)

On peut utiliser les function sans avoir recour à les déclarer, c'est-à-dire sans utiliser de prototypes. C'est uniquement les corps de ces function qui sont utilisés. Si ces corps sont implantés dans la partie déclarative de l'architecture, les function concernées sont locales à ces architectures. Si ces corps sont implantés au niveau de l'entity, les function concernées sont visibles par toutes les architectures de cette entity. Cette technique est utilisée si les corps des fonctions ne sont pas grands.

```
-- File My_Package.vhd

library ieee ;
use ieee.std_logic_1164.all;

package My_Package is

    subtype Small_Integer is  integer range 0 to 10;

end package;


-- File Min_Max.vhd

library ieee;
use ieee.std_logic_1164.all;

library work;
use work.My_Package.all;

entity Min_Max is
    port(
        a, b, c       : in    Small_Integer;

        Max,  Min : out Small_Integer;
    );

end Min_Max_Functions;

architecture Arch of Min_Max is

    function Min_Value (x, y, z : in integer) return integer is

        variable temp: Small_Integer;

    begin
        if x < y then
            temp := x;
        else
            temp := y;
        end if;

        if temp < z then
            return temp;
        else
            return z;
        end if;
    end Min_Value;


    function Max_Value(x,y : in integer) return integer is
```

```
        variable temp: Small_Integer;
    begin                    >
        if x > y then
            temp := x;
        else
            temp := y;
        end if;
        if temp > z then
            return temp;
        else
            return z;
        end if;
    end Max_Value;
begin
1.  Proc: process(a, b, c)
2.  begin
3.      Max  <= Max_Value(a, b, c);
4.      Min  <= Min_Value(a, b, c);
5.  end process Proc;
6.  Max  <= Max_Value(a, b, c);
7.  Min  <= Min_Value(a, b, c);
end Arch;
```

Les deux fichiers ci-dessus doivent être assemblés avant la compimlation. Notez que les function sont invoquées dans le coté droit de l'opérateur d'affectation <=. Cette invocation peut être faite soit de façon séquentielle, selon les instructions 1-5, ou de façon concurrente, selon les instructions 6-7.

Dans l'une comme dans l'autre de ces deux techniques, le circuit généré est bien évidemment le même. Il est donné dans la figure ci dessosu.

## II. LES PROCEDURES : procedure

### II.1. DEFINITION

Pareil que pour les fonctions, VHDL nous permet de rassembler un ensemble d'instructions sous forme de sous programmes appelés procedures (procedure). Elles peuvent être appelées de différents endroits, et elles sont utilisées pour améliorer la lisibilité et exploiter la possibilité de réutilisation du code VHDL.

Une procedure peut être déclarée dans la partie déclarative d'une architecture ou un package et ses paramètres peuvent être exclusivement un signal, une variable ou une constant. Elle peut avoir plusieurs entrées et plusieurs sorties et ses paramètres peuvent être en mode in, out et inout.

L'instruction return peut également être utilisée dans les procédures cependant, à la différence des fonctions, cette instruction ne devrait retourner aucune valeur.

Une procedure peut retourner implicitement des valeurs à travers ses paramètres formels ou à travers les variables globales. Les variables décarées au sein d'une procedure, sont locales à celle-ci et un contrôle du temps est autorisé (contain timing : un wait) sauf si elle est appelée par un process avec une liste de sensibilité ou depuis une function;

Une procedure peut avoir un corps uniquement ou avoir un corps et une déclaration (prototype). Quand la procedure est fournis dans un package, sa déclaration est placée dans la déclaration du package et son corps est placé dans le corps (body) de celui ci.

☞ VHDL autorise la surcharge des procedure.

☞ Une procédure ne devrait jamais être figurée dans le coté droit de l'opérateur d'affectation <=.

### II.2. COMPOSANTES D'UNE PROCEDURE

Une procedure peut être composée, si elle est déclarée dans un package, de deux parties: Une partie déclarative, appellé également prototype, qui ne comprend que le nom de la procedure et les noms et les types de ses paramètres formels et une partie corps (body), qui comprend le détail de ce que fait la procedure selon la syntaxe suivante::

```
procedure Procedure_Identifier [(List of formal parameters)] is
    Zone de déclarations
begin
        Zone d'instructions séquentielles
end nom;
```

où :

Procedure_Identifier : désigne le nom de la procedure

List of formal parameters : spécifie les paramètres d'entrée de la procédure. Ils ont la forme :

Parametre_class Parametre_Identifier : Parametre_mode Parametre_type

où Parametre_class peut être un signal (signal), une constante (constant ) ou une variable (variable). Parametre_Identifier désigne le nom du paramètre. Parametre_mode peut être in, out ou inout. Parametre_type peut être std_logic, integer,....

Exemple 1 :

```
Procedure Min_Values ( signal a, b : in integer; signal c : out integer) is
Begin
    if a < b then
```

```
        c <= a ;
    else
        c<= b ;
    end if;
end Min_values;
```

### II.3. UTILISATION DES PROCEDURES

Une procedure (comme une function) peut être utilisée selon 3 méthodes (comme pour les component)

1. Décrites dans la zone déclarative de l'architecture ou l'entity qui l'utilisent: seul le corps de la procédure est donné (il n y a pas de déclaration);
2. Déclarée et décrite dans un package : La déclaration de la procédure est faite dans la partie déclarative du package, et le corps de la procédure est donné dans le corps (body) du package.
3. Décrite dans un fichier (dans une Library quelconque) : Ici il faut déclarée la procedure dans la zone déclarative de l'architecture qui l'utilise.

☞ Peuvent être utilisés dans le monde concurrent ou séquentiel.

Ci-dessous ne sera décrit que la première technique.

### II.4.1. UTILISATION DE PROCEDURES SANS DECLARATION (SANS PROTOTYPE)

Ici, c'est uniquement le corps de la procedure qui est utilisé. Dans l'exemple ci-dessous, les corps de deux procedure, Max_Value et Min_Value, sont donnés dans la partie déclarative de l'architecture.

Exemple : Determination of the maximum and minimum of three parameters

```
-- File My_Package.vhd

library ieee ;
use ieee.std_logic_1164.all;

package My_Package is

    subtype Small_Integer is  integer  range 0 to 10;

end package;

-- File Min_Max.vhd

library ieee;
use ieee.std_logic_1164.all;

library work;
use work.My_Package.all;

entity Min_Max is
    port(
        a, b, c  :  in     Small_Integer;
```

```
        Max    : out   Small_Integer;
        Min    : out   Small_Integer
    );
end Min_Max;

architecture Arch of Min_Max is
    procedure Min_Value (x, y, z: in Small_Integer; signal Minimum : out Small_Integer) is
        variable temp: Small_Integer;
    begin
        if x < y then
            temp := x;
        else
            temp :=y;
        end if;
        if temp < z then
            Minimum <= temp;
        else
            minimum <= z;
        end if;
    end Min_Value;
    procedure Max_Value(x, y, z: in Small_Integer; signal Maximum : out Small_Integer) is
        variable temp: Small_Integer;
    begin
        if x > y then
            temp:=x;
        else
            temp:=y;
        end if;
        if temp > z then
            Maximum <= temp;
        else
            Maximum <= z;
        end if;
    end Max_Value;
begin
1.  Proc: process(a, b, c)
2.  begin
3.      Max_Value(a, b, c, Max);
4.      Min_Value(a, b, c, Min);
5.  end process Proc;
6.  Max_Value(a, b, c, Max);
7.  Min_Value(a, b, c, Min);

end Arch;
```

Les deux fichiers ci-dessus doivent être assemblés avant la compimlation. Ceci permettera de rendre utilisable le contenu du Package.

L'invocation d'une structure peut être exécutée de façon séquentielle, comme le préciusent les instructions 1-5, ou de façon concurrente, commele précisent les instructions 6-7.

☞ tion des bibliothèques.