VHDL Récapitulatif

1. Structure entity / architecture

Un composant VHDL est constitué d'une description interne : **entity** et d'une description fonctionnelle : l'***architecture***

```
entity ma_fonction is
        port ( - - déclaration des signaux externes
                nom : mode type ;
        ) ;
end ma_fonction ;

architecture arch_ma_fonction of ma_fonction is
        -- Déclaration des signaux et constantes
begin
        -- Description du fonctionnement interne
end arch_ma_fonction ;
```

2. Port

Les signaux d'interface d'une entity constituent les ports. L'ordre de déclaration des signaux doit être repris par toute description utilisant le composant.

3. Modes

4 modes définissent les sens des signaux :

- **in** : applicable à un signal d'entrée (monodirectionnel)
- **out** : applicable à un signal de sortie (monodirectionnel)
- **inout** : applicable à un signal d'entrée/sortie (bidirectionnel)
- **buffer** : applicable à un signal de 'sortie/entrée' (monodirectionnel) {éviter son utilisation}

4. Types

4.1. Types prédéfinis :

integer : type scalaire entier définissant des entiers compris entre $-(2^{31}-1)$ et $2^{31}-1$

l'intervalle peut être limité :

*num : **in** integer **range** -128 to 127*

➔ permet de ne pas utiliser 32 bascules

natural : sous type du type integer limité aux nombres positifs ou nuls

positive : sous type du type integer limité aux nombres positifs

bit : type scalaire énuméré, prenant les valeurs '0' ou '1'

bit_vector : ensemble de bit

*D : **in** bit_vector(0 **to** 7)*

*D : **in** bit_vector(7 **downto** 0)*

l'indice de gauche définit le bit de poids fort et l'indice de droite celui de poids faible.

4.2. Types complémentaires (accessibles via la bibliothèque ieee)

std_logic ou **std_logic_vector** : extension du type bit et bit_vector pouvant prendre 9 valeurs :

'U' : non initialisé
'X' : niveau inconnu, forçage fort
'0' : niveau 0, forçage fort
'1' : niveau 0, forçage fort
'Z' : haute impédance
'W' : niveau inconnu, forçage faible
'L' : niveau 0, forçage faible
'H' : niveau 1, forçage faible
'-' : quelconque

Remarque : Il est fortement recommandé d'utiliser le type std_logic de la bibliothèque ieee en remplacement du type bit. Pour un synthétiseur logique les valeur '0' et 'L' sont équivalent de même que '1' et 'H' , les valeurs 'U', 'X' et 'W' ne sont pas utilisées.

2 types complémentaires pour les traitements arithmétiques :

signed *: type signed is array (natural range <>) of std_logic ;*

unsigned *: type unsigned is array (natural range <>) of std_logic ;*

la distinction réside dans l'action différenciées des opérateurs.

4.3. Types composites:

***array** : regroupement d'objet de même type*
***type** table8x4 **is array**(0 **to** 7, 0 **to** 3) **of** bit;*
***record :** regroupement d'objet de type différent*
type exemple **is** record
X,Y : std_logic;
end record;

4.4. Types définis par l'utilisateur :

type ETAT is (Repos, Init, Activ, attente, Fin, Erreur);
MON_ETAT : buffer ETAT;

5. Objets

- **signal** : signaux internes (les ports sont également des signaux)
  Architecture ...
  signal nom : type;
  ***begin** ...*
- **constant** : signal interne auquel est associée une valeur
  Architecture ...
  constant nom : type := valeur;
  ***begin** ...*
- **variables** : objet contenant une donnée temporaire (déclarable uniquement à l'intérieur ***process***)
  process …
  variable result : st_logic := '0';
  begin
- **alias** : permet de dénommer un objet de différentes manières
  signal dbus : bit_vector(15 **downto** 0);
  alias octet1 : bit_vector(7 **downto** 0) **is** dbus(15 **downto** 8);
  alias octet2 : bit_vector(7 **downto** 0) **is** dbus(7 **downto** 0);

6. Généralités sur l'écriture

- **Commentaires :**
  débutent par -- et se poursuivent jusqu'en fin de ligne
- **Majuscules et minuscules :**
  pas de distinction
- **Valeurs littérales :**
  - entier : au plus simple (123, -93)
  - bit : entre quotes (‘ 1 ’ ou ‘ 0 ’)
  - vecteurs de bits : entre guillemets ("1001")
  - caractère : entre quotes (‘ A ’)
  - chaîne de caractères : entre guillemets ("ABCD")
- **Instructions :**
  possibilité de les étendre sur plusieurs lignes
  se terminent par un point virgule (**;)**
- **Assignation de vecteurs**
  signal A : std_logic_vector (7 downto 0); - -bus 8 bit
  - Assignation d'une valeur à un des éléments du vecteur : utilisation du nom de vecteur suivit du numéro de l'élément
    A(3) <= '1'
  - Assignation d'une valeur à l'ensemble du bus
    A <= "1011 0111";
    A <= (7 => '1', 6 =>0, 5 downto 4 => '0', 3 => '1', others => '1');

7. Modularité

Les composants créés peuvent être réutilisés dans d'autres descriptions pour cela, ces derniers doivent être rendus visibles par déclaration du composant :

- déclaration dans la nouvelle description
- déclaration dans un "package" réutilisable

7.1. Déclaration dans l'architecture

pour pouvoir réutiliser un composant, il est nécessaire de le déclarer en tant que composant :
architecture** a **of** nouvelle_description **is
***component** basculeD*

```
port (
        --déclaration des signaux de l'entity du composant à réutiliser
);
end component;
-- déclaration des signaux et constantes
begin…
```

7.2. Librairie et paquetage

Constitution de librairies et packages :
les déclarations des components peuvent être regroupés ensemble dans un paquetage

```
package nom_package is
-- déclaration des composants du paquetage
component basculeD is    -- même nom que l'entité auquel le composant fait référence
port (
        --déclaration des signaux
);
end component;
...
end nom_package;
```

Lorsque les paquetages sont regroupés dans des fichiers placés dans un répertoire ils constituent une librairie dont l'emplacement et le nom devront être spécifié au synthétiseur.

Déclaration :

```
library ieee ;      -- nom de la librairie
use ieee.std_logic_1164.all ;      -- accès au package std_logic_1164 de la librairie ieee
```

Librairies particulières (maxplus2 uniquement) :
Pour les opérations arithmétiques sur des types std_logic, signed, unsigned :

- paquetage ***std_logic_1164*** : définit le type std_logic
- paquetage ***std_logic_arith*** : définit les type unsigned et signed et les surcharge d'opérateur pour ces types
- paquetage ***std_logic_unsigned*** et ***std_logic_signed*** permettent d'utiliser les type std_logic_vector comme s'ils étaient des types unsigned ou signed

Conversions de types (pakage std_logic_arith) :

CONV_INTEGER(valeur)	*Convertie un type std_logic ou un/signed en entier*
CONV_UNSIGNED(valeur, taille en bits)	*convertit un type entier, std_logic ou signed en unsigned*
CONV_SIGNED(valeur, taille en bits)	*convertit un type entier, std_logic ou unsigned en signed*
CONV_STD_LOGIC_VECTOR(valeur, taille en bits)	*convertit un type entier ou un/signed en std_logic_vector*
UNSIGNED (valeur) ou SIGNED (valeur)	*convertit un type std_logic_vector en type unsigned ou signed*

7.3. Instanciation des composants

L'utilisation du composant dans une architecture s'effectue de la manière suivante :
U0 : ma_fonction ***port map*** *(a, b,c) ;* les signaux doivent être passés en paramètres dans l'ordre de déclaration des signaux de l'entity.

Lorsque certain signaux ne sont pas utilisé mettre

- un signal constant pour les signaux d'entrée
- l'expression "open" pour les signaux de sortie

lorsque les signaux ne sont pas appelé dans l'ordre utiliser l'association par nom **port map** (y=>b, z=>c,x=>a)
U0 est le nom de l'instance du composant *ma_fonction (ce nom n'a pas besoin d'être déclaré)*

7.4. paramètres générique

Les composants peuvent être déclarés avec des paramètres génériques
Ces paramètres génériques permettent d'instancier des composants avec des tailles variables

```
entity MUX_2vers1 is
generic (largeur : integer :=8); --paramètre par défaut
port (
        selecteur : in std_logic
        A,B : in std_logic_vector(largeur downto 0)
        C : out std_logic_vector(largeur downto 0) );
end MUX;
architecture a of MUX is
begin
C <= A when selecteur = ' 0 ' else B;
end a;
```

-- instanciation

U0 : MUX ***generic map*** *(12)* ***port map*** *(Sel, Din1, Din2 Dout);*

8. Fonctionnements concurrents et séquentiels

VHDL se distingue des langages de programmation traditionnels par son fonctionnement concurrent ou parallèle : les instructions, sauf exception, sont évaluées en permanence et simultanément

A <= B;

B <= C;

A tout moment A, B et C ont la même valeur.

Process

- Déclenchement sur changement d'un des signaux sensibles.
- Instructions évaluées séquentiellement
- Modifications prises en compte à la fin du processus (excepté pour les variables qui prennent leur valeur immédiatement)

nom_optionnel : ***process*** *(liste des signaux sensibles)*

{partie déclarative du process (déclaration des variables)}

begin

{instructions mode séquentiel}

end process *nom_optionnel*

9. Opérateurs

9.1. Opérateurs logiques

Opérateurs	type des opérandes	signification
and	boolean bit bit vector	et
nand		non-et
or		ou
nor		non-ou
xor		ou exclusif
xnor		égal ou nonou exclusif
not( )		non

9.2. Opérateurs relationnels

Opérateurs	type des opérandes	signification
=	tout type scalaire sortie : booléen	égal
/=		non égal
<		inférieur
<=		inférieur ou égal
>		supérieur
>=		supérieur ou égal

9.3. Opérateurs arithmétiques

Opérateurs	type des opérandes	signification
+	entiers réels	addition
-		soustraction
*		multiplication
/		division
abs		valeur absolue
**		exponnentiel
mod	entier	modulo
rem		reste

Remarque : Les opérateurs autres que les opérateurs d'addition et de soustraction ne sont jamais utilisés en synthèse logique. Certains outils de synthèse acceptent néanmoins l'opération de multiplication.

9.4. Opérateurs décalage (peu utilisé)

Opérateurs	type des opérandes	signification
sll	bit vector	logique gauche
srl		logique droit
rol		circulaire gauche
ror		circulaire droit

9.5. Opérateur concaténation

Opérateurs	type des opérandes	signification
&	bit, bit vector	concaténation

A & B ⇔ "AB"
A & B & others => '0' ⇔ "AB000000" si vecteur 8 bits
Buffer <= buffer(5 downto 0) & "00" – décalage logique à gauche de 2 bits

9.6. Surcharge d'opérateur

Les opérateurs sont définis pour des types de signaux précis. Ils peuvent cependant être surchargés (overloaded) de manière à pouvoir être utilisés avec des signaux autres que ceux pour lesquels ils ont été définis.

⇒ Utilisation des packages tels que ceux de la librairie IEEE pour les signaux du type std_logic : '**use** ieee.std_logic_arith.all' ou '**use** ieee.std_logic_unsigned.all'.

10. Instructions

10.1. Mode concurrent

- Assignation inconditionnelle : **<=**
- Assignation conditionnelle : *signal <= expression **when** conditions **else** expression;*
- Assignation sélective :
```
with sélecteur select
    signal <=   expression when valeur_sélecteur ,
                expression when valeur_sélecteur ,
                expression when others ;
```
- Instruction **generate** :
```
étiquette : for variable_boucle in val_début to (ou downto) val_fin
        generate
            {if condition then generate
                {instructions concurrentes}
            end generate}
        end generate etiquette;
```

10.2. Mode séquentiel (process)

- Assignation inconditionnelle de signaux : **<=**
- Assignation inconditionnelle de variables : **:=**
- Assignation conditionnelle :
```
if condition then instructions ;
elsif conditions then instructions ;
else instructions ;
endif ;
```
- Assignation sélective :
```
case sélecteur is
        when valeur_sélecteur => instructions ;
            when valeur1_sélecteur | valeur2_sélecteur | … => instructions ;
            when others => instructions ;
end case ;
```

- Boucle **for … loop** :
```
[étiquette :] for variable-boucle in val-début to (ou downto) val-fin loop
        instructions ;
end loop [étiquette] ;
```
- Boucle **while … loop** :
```
[étiquette :] while condition loop
        instructions ;
end loop [étiquette] ;
```
- Instruction **next** et **exit** : l'instruction **next** permet de terminer une itération de boucle sans exécuter les instructions suivantes, l'instruction **exit** permet de terminer l'exécution de la boucle.

Remarque : les instructions de boucle sont très utilisées en simulation (description de stimuli). En synthèse, elles doivent être manipulées avec précaution (toujours essayer de se mettre à la place du synthétiseur).

Exemple :
```
validation : for i in 0 to 3 generate
        aval(i) <= a(i) and val;
```
➔ les boucles sont souvent utilisées pour simplifier l'écriture (instructions répétitives)

11. Attributs

- propriétés associées à un objet
- s'utilisent accolés à un signal (exemple : clock***'event)***

Exemples ***signal*** *indice* ***is*** *integer* ***range*** *0* ***to*** *255;*
signal *octet* ***is*** *std_logic_vector (7* ***downto*** *0);*
signal *nible* ***is*** *std_logic_vector (0* ***to*** *3);*

- left : rend la valeur la plus à gauche de l'intervalle
  *indice**'left** : 0 ;* *octet**'left** : 7 ;* *nible**'left** : 0*
- right : rend la valeur la plus à droite de l'intervalle
  *indice**'right** : 255 ;* *octet**'right** : 0 ;* *nible**'right** : 3*
- high : rend la valeur la plus élevé de l'intervalle
  *indice**'high** : 255 ;* *octet**'high** : 7 ;* *nible**'high** : 3*
- low : rend la valeur la moins élevé de l'intervalle
  *indice**'low** : 0 ;* *octet**'low** : 0 ;* *nible**'low** : 0*
- lenght : rend le nombre d 'élément de la liste
  *indice**'lenght** : 255 ;* *octet**'lenght** : 8 ;* *nible**'lenght** : 4*
- range : rend l'intervalle de variation d'un signal dimensionné
  *octet**'range** : 7* ***downto*** *0 ;* *nible**'range** : 0* ***to*** *4*

possibilité de définir des attributs utilisateur