VHDL Avancé 2 : Concepts et syntaxes avancés en VHDL

Clément Foucher

VHDL avancé

CONCEPTS ET SYNTAXES AVANCÉS EN VHDL

BUT GEII Toulouse S6 ESE

Composants génériques

HAUTE IMPÉDANCE

ASSERTIONS

GENERATE

Notion de généricité

- En VHDL, il est possible de rendre un composant générique
- Un composant générique dispose de paramètres dont la valeur est définie à l'instanciation
- Cela signifie que qu'un même composant peut avoir plusieurs instances qui ne seront pas identiques entre elles si les valeurs fournies pour les paramètres génériques sont différentes

Syntaxe de l'`entity`

- La généricité se déclare dans la description de l'`entity`
- On ajoute un nouveau bloc en plus du bloc `port` : le bloc `generic`
- La syntaxe de ce bloc est la même que celle du bloc `port`
  - On déclare `nom : type`
    - Mais pas d'indication de sens (in/out)
  - Différents paramètres sont séparés par des « ; »
  - Il faut placer un « ; » après la parenthèse fermante du bloc

```
entity generic_cpt is
generic(size : integer);
port(clk    : in  std_logic;
     reset  : in  std_logic;
     en     : in  std_logic;
     Q      : out std_logic_vector(size-1 downto 0)
    );
end entity;
```

Notez qu'ici, la taille du vecteur de sortie dépend du paramètre générique

Adaptation du code

- Les paramètres génériques peuvent être utilisés comme n'importe quelle constante
- On peut par exemple utiliser un paramètre de type `integer`
  - Dans un range de signal
  - Dans une boucle (voir `for…generate` plus loin)

```
architecture ar of generic_cpt is
    Qi : unsigned(size-1 downto 0);
begin

    process(clk, reset)
    begin
        if reset = '1' then
            Qi <= (others => '0');
        elsif rising_edge(clk) then
            if en = '1' then
                Qi <= Qi + 1;
            end if;
        end if;
    end process;

    Q <= std_logic(Qi);

end architecture;
```

Instanciation d'un composant générique

- L'instanciation d'un composant générique ajoute une section `generic map` dans laquelle on définit les valeurs des paramètres
- Pas de « ; » après la parenthèse, celui-ci se place toujours à la fin de l'instruction complète

```
architecture ar of monCompo is

begin

     u_cpt:entity work.generic_cpt(ar)
       generic map(size => 32)
       port map(clk   => clk,
                reset => reset,
                en    => en,
                Q     => Q);

end architecture;
```

Valeur par défaut

- Il est possible de définir une valeur par défaut pour un générique
- Si une valeur par défaut est présente, on peut omettre le `generic map` pour ce paramètre, qui prendra alors sa valeur par défaut

```
entity generic_cpt is
generic(size : integer := 32);
port(clk   : in  std_logic;
     reset : in  std_logic;
     en    : in  std_logic;
     Q     : out std_logic_vector(size-1 downto 0)
    );
end entity;
```

```
u_cpt:entity work.generic_cpt(ar)
  port map(clk   => clk,
           reset => reset,
           en    => en,
           Q     => Q);
```

Instancie un compteur sur 32 bits

```
u_cpt:entity work.generic_cpt(ar)
  generic map(size => 8)
  port map(clk   => clk,
           reset => reset,
           en    => en,
           Q     => Q);
```

Instancie un compteur sur 8 bits

Types communs pour les paramètres génériques

- Tous les types VHDL sont autorisés en tant que paramètre générique, mais on trouve le plus souvent :
  - Des `integer` pour exprimer une taille de vecteur
  - Des `boolean` pour activer ou désactiver une fonctionnalité
  - Des `std_logic` pour indiquer le niveau actif d'un reset

Exemples

- Choix du niveau logique du reset

```
entity generic_cpt is
generic(size        : integer;
        reset_value : std_logic;
       );
port(clk   : in  std_logic;
     reset : in  std_logic;
     en    : in  std_logic;
     Q     : out std_logic_vector(size-1 downto 0)
    );
end entity;

architecture ar of generic_cpt is
     Qi : unsigned(size-1 downto 0);
begin

     process(clk, reset)
     begin
          if reset = reset_value then
               Qi <= (others => '0');
          elsif rising_edge(clk) then
               if en = '1' then
                    Qi <= Qi + 1;
               end if;
          end if;
     end process;

     Q <= std_logic(Qi);

end architecture;
```

COMPOSANTS GÉNÉRIQUES

Haute impédance

ASSERTIONS

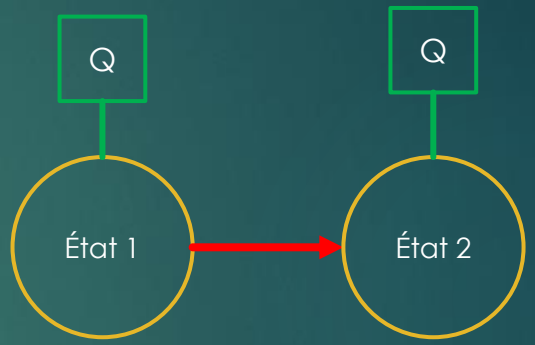
GENERATE

Valeurs possibles d'un `std_logic`

- Outre les valeurs 0 et 1, un `std_logic` peut prendre les valeurs suivantes :
  - `'H'`, `'L'` (high et low)
  - `'Z'` (haute impédance)
  - `'W'` (weak)
  - `'U'` (non initialisé)
  - `'X'` (valeur inconnue)
  - `'-'` (valeur sans importance)
- Toutes ces valeurs ne sont pas utilisables dans tous les contextes
  - Exemple : `'U'` n'a de sens qu'en simulation
- Notamment, `'Z'` ne peut être utilisé que si des buffers 3 états existent dans le FPGA
- Ces buffers sont généralement disponibles sur les pattes d'entrée-sortie d'un FPGA

Utilisation de l'état haute impédance

- Rappel : notion de *driver*
  - En VHDL, un *driver* est une source d'état logique
  - Dans le cas normal, il est absolument impossible d'avoir plusieurs drivers pour un même signal
- Exemple



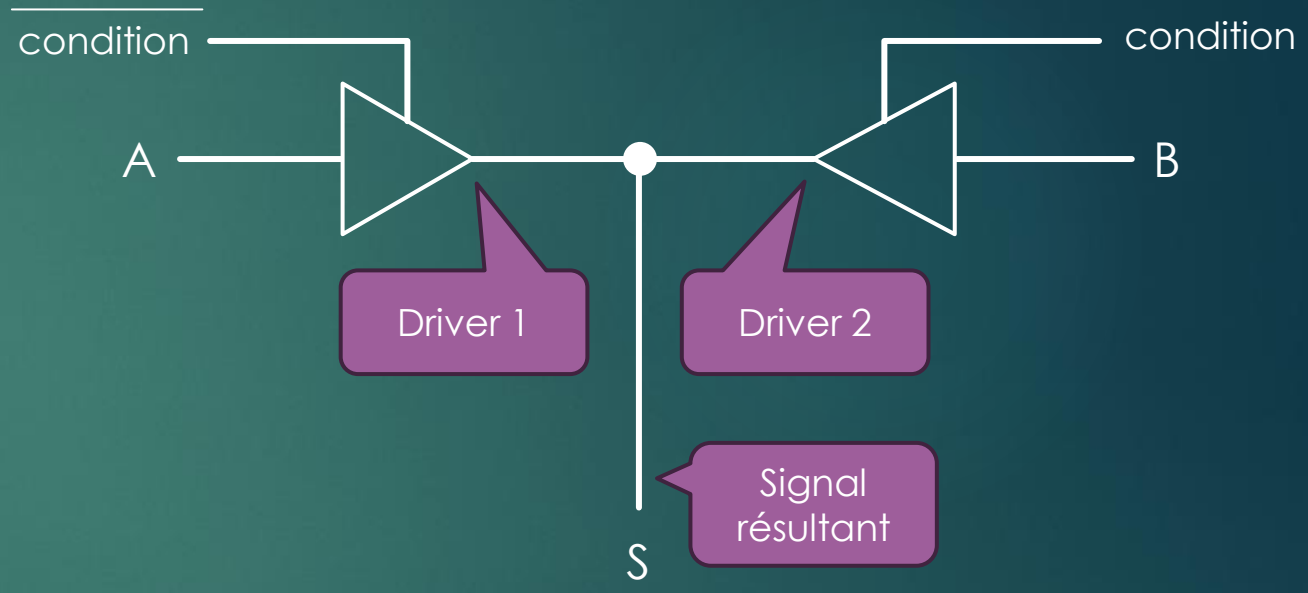

```
-- Erreur courante dans les MAE
Q <= '1' when etat_courant = etat_1 else '0';
Q <= '1' when etat_courant = etat_2 else '0';
```

Si on est dans l'état 1 ou l'état 2, l'une de ces lignes donne la valeur `'0'` à Q, et l'autre la valeur `'1'`
➔ court-circuit

En simulation, on aura l'état `'X'` dans le `std_logic`

Utilisation de l'état haute impédance

- Toutefois, il existe un cas dans lequel on peut avoir plusieurs drivers pour le même signal : si ceux-ci sont protégés par des buffer 3 états.
  - Mais il faut toujours faire attention à ce qu'il y ait en permanence au moins un buffer 3 états en état haute impédance ➔ condition mutuellement exclusive
- Les buffers 3 états ne sont en général pas utiles à l'intérieur d'un FPGA (on préfèrera utiliser des multiplexeurs)
- Mais ils peuvent l'être pour attaquer un signal externe
  - Exemple que vous connaissez bien : bus I2C




```
S <= A when condition = '0' else 'Z';
S <= B when condition = '1' else 'Z';
```

COMPOSANTS GÉNÉRIQUES

HAUTE IMPÉDANCE

Assertions

GENERATE

Concept d'assertion

- Une assertion sert à vérifier qu'une condition est réalisée, et à défaut d'afficher un message d'avertissement ou d'erreur, voire d'interrompre une exécution
- En VHDL, cela n'a de sens qu'en simulation ou pendant la synthèse, car ensuite on n'a pas de code mais un circuit logique
  - Ils peuvent notamment être utilisés dans un testbench

Syntaxe de l'assertion

- Une assertion est constituée
  - D'une condition à vérifier
  - D'un message à afficher si la condition n'est pas respectée
  - D'un niveau de sévérité

```
assert condition report "message" severity error;
```

Niveaux de sévérité

- Les niveaux disponibles sont
  - note
    - Simple information à afficher
  - warning
    - Affiche un message d'avertissement mais sans interrompre la simulation
  - error
    - Affiche un message d'erreur et interrompt la simulation
  - failure
    - Affiche un message d'erreur et interrompt la synthèse

COMPOSANTS GÉNÉRIQUES

HAUTE IMPÉDANCE

ASSERTIONS

Generate

Principe de base du `generate`

- Le mot-clé `generate` permet de rendre conditionnel la génération d'une partie du code
- Il est généralement dépendant de la valeur d'un paramètre générique
- Si une condition est remplie (`if … generate`), le contenu du bloc sera synthétisé, sinon il n'existera pas
  - Équivalent du `#if` ou `#ifdef` en C
- Il permet également de générer un nombre variable de structures du même type (`for … generate`)

if … generate

- Exemple : `if … generate`
- Notez qu'un generate doit obligatoirement être nommé
- Attention : le `else` dans un `generate` n'existe que depuis le VHDL 2008
  - Dans les versions antérieures, il faut deux blocs `if … generate` avec deux conditions opposées

```
entity generic_cpt is
generic(size         : integer;
        buffer_input : std_logic;
       );
port(clk    : in  std_logic;
     reset  : in  std_logic;
     en     : in  std_logic;
     Q      : out std_logic_vector(size-1 downto 0)
    );
end entity;

architecture ar of generic_cpt is
     Qi : unsigned(size-1 downto 0);
     en_i : std_logic;
begin

     buffer:if buffer_input = true generate
          process(clk, reset)
          begin
               if reset =  '1' then
                    en_i <= '0';
               elsif rising_edge(clk) then
                    en_i <= en;
               end if;
          end process;
     else
          en_i <= en;
     end generate;

     […]
end architecture;
```

for … generate

```
package array_pkg is
    type std_array      is array(natural range <>) of std_logic_vector(7 downto 0);
    type unsigned_array is array(natural range <>) of unsigned(7 downto 0);
end package;
```

- Exemple : `for … generate`
- Notez qu'un generate doit obligatoirement être nommé
- Notez ici la définition, dans un package, de types spécifiques : `array`
  - Les `array` (tableaux) permettent de regrouper plusieurs signaux du même type pour en faire un vecteur
  - Le `range <>` permet d'indiquer que la taille sera définie à l'utilisation

```
entity multi_cpt is
generic(number_of_cpt : integer
       );
port(clk    : in  std_logic;
     reset  : in  std_logic;
     en     : in  std_logic_vector(number_of_cpt-1 downto 0);
     Q      : out std_array(number_of_cpt-1 downto 0)
    );
end entity;

architecture ar of generic_cpt is
     Qi : unsigned_array(number_of_cpt-1 downto 0);
begin

     counters:for cptnum in 0 to number_of_cpt-1 generate
          process(clk, reset)
          begin
               if reset = reset_value then
                    Qi(cptnum) <= (others => '0');
               elsif rising_edge(clk) then
                    if en(cptnum) = '1' then
                         Qi(cptnum) <= Qi(cptnum) + 1;
                    end if;
               end if;
          end process;

          Q(cptnum) <= std_logic(Qi(cptnum));
     end generate;

end architecture;
```

L'icône d'information indique que le contenu de ces slides est à titre informatif, il ne vous est pas demandé de le retenir

Bonus

Tableaux de tableaux

- Peut-on faire un `array` de `std_logic_vector` dont la taille est définie à l'utilisation ?
  - Oui… mais attention à la syntaxe !

```
package array_pkg is
    type std_array      is array(natural range <>) of std_logic_vector;
    type unsigned_array is array(natural range <>) of unsigned;
end package;
```

Pas de range défini

```
entity multi_cpt is
generic(number_of_cpt : integer;
        cpt_size      : integer
       );
port(clk    : in  std_logic;
     reset  : in  std_logic;
     en     : in  std_logic_vector(number_of_cpt-1 downto 0)(cpt_size-1 downto 0);
     Q      : out std_array(number_of_cpt-1 downto 0)(cpt_size-1 downto 0)
    );
end entity;
```

Définition du range de l'`array`

Définition du range du `std_logic_vector`