Unités : CSN & SysLog2

Le langage VHDL pour la synthèse visseries et astuces

HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Etienne Messerli

Septembre 2021

Visseries et astuces

- Voici des éléments très pratique pour les descriptions VHDL :
  - opérateur de concaténation "&"
  - notation par agrégat et mot clé others
  - affectation en hexadécimal
  - les attributs les plus courants
  - description d'une porte 3 états
  - comparaison de vecteur et pièges

L'opérateur de concaténation "&"

- Exemples de regroupement pour affecté un vecteur :

```
vect4 <= D & C & B & A;
vect8 <= vect10(9 downto 6) & A & B & "00";
```

- Exemples de décomposition de vecteurs :

```
vect4 <= vect8(5 downto 2);
signal <= vect8(4);
vect8(5 downto 3) <= "010";
```

Affectation par notation d'agrégat

- La notation par agrégat permet d'indiquer la valeur d'un type composite, comme par exemple les tableaux (array). Cela est très utile pour l'affectation des vecteurs (type array). Voici des exemples :

```
--Soit la déclaration suivante:
signal vecteur : std_logic_vector(3 downto 0);

--Nous pouvons affecter le vecteur comme suit :
vecteur <= ('1','0','0','1') ; --idem <="1001";
```

Agrégat et mot clé others …

Le mot clé "others" permet d'indiquer que tous les autres éléments de l'agrégat sont affectés par une même valeur.

Si le mot clé "others" est utilisé seul, cela permet d'affecter TOUS les éléments avec la même valeur.

Exemples :

```
vect_A <= (others => '0'); --tout à 0
vect_C <= (others => 'Z'); --tout à Z

--ou affecté vecteur avec le même signal my_signal
vect_4 <= (others => my_signal);
```

… agrégat et mot clé others

La notation par agrégat permet d'indiquer le numéro de l'élément affecté. Combiné avec le mot clé "others", cela permet des affectations très pratique.

Exemples :

```
--affectation : MSB à '1', autres bits à '0'
vect8 <= (7 => '1', others => '0');

--affectation : MSB à '0', autres bits à '1'
vect8 <= (7 => '0', others => '1');

--Affectation : LSB à '1', autres bits à '0'
vect8 <= (0 => '1', others => '0');
```

Affectation en hexadécimal …

- Utilisation de valeurs en hexadécimal
- Exemple d'affectation de vecteurs:

```
vecteur_8bits  <= x"6A";
vecteur_24bits <= x"2A_4F5C"
vecteur_4bits  <= x"8";
```

- Attention valable uniquement pour des vecteurs ayant un multiple de la base 2, soit : 4, 8, 12, 16 ...

… affectation en hexadécimal

- Cas de vecteurs de longueur non multiple de 4 :

Déclarer une constante ayant la taille multiple de 4 supérieur

```
constant cst16 : std_logic_vector(15 downto 0)
                                        := x"2A04";
```

puis prendre que le range nécessaire

```
vect14bits <= cst16(13 downto 0);
```

Ou, compléter par des bits la partie non multiple de 4

```
vect14bits <= "10" & x"A04";
```

Attributs prédéfinis pour tableaux (array) ...

- Ces attributs sont indispensables pour rendre les descriptions paramétrables en association avec la concaténation & et la notation par agrégat.
  - permettent de manipuler des array, comme par exemple les vecteurs : `std_logic_vector(7 downto 0)`
  - nécessaires pour rendre les descriptions paramétrables, donc indépendantes de la taille des tableaux (vecteurs)
  - utilisable dans tous les types de description: synthèse, spécification ou test-bench

… les attributs pour les array…

- Voici les principaux attributs pour les array:

'left	indice de gauche
'right	indice de droite
'high	indice supérieur (MSB)
'low	indice inférieur (LSB)
'length	longueur du tableau (array)
'range	intervalle des indices
'reverse_range	intervalle inverse des indices

… les attributs pour les array

- Exemple d'utilisation des attributs sur un signal:

```
signal data : std_logic_vector(7 downto 0);
```

dans ce cas :	correspond à
`data'high = data'left`	`7`
`data'low  = data'right`	`0`
`data'length`	`8`
`data'range`	`7 downto 0`
`data'reverse_range`	`0 to 7`

Exemples utilisation attributs array …

- Déclaration d'un signal interne :

```
signal compteur: std_logic_vector(7 downto 0);
signal cpt_int: unsigned(compteur'range);
```

- Décalage à droite :

```
vect_shr <= '0' & vecteur(vecteur'high downto 1);
```

- Rotation à gauche :

```
vect_rol <= vect(vect'high-1 downto 0) &
                                    vect(vect'high);
```

- Initialiser un vecteur signé à la valeur min

```
vect <= (vect'high => '1', others => '0');
```

Porte trois états (three states)

Description avec une instruction concurrente :

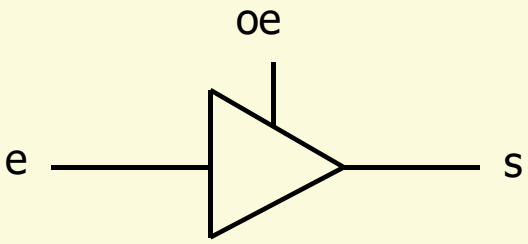
```
library ieee;
use ieee.std_logic_1164.all;

entity trois_etats is
  port (e_i, oe_i : in std_logic;
        s_o : out std_logic;);
end trois_etats;

architecture flot_don of trois_etats is
begin

  s_o <= e_i  when oe_i = '1' else 'Z';

end flot_don;
```



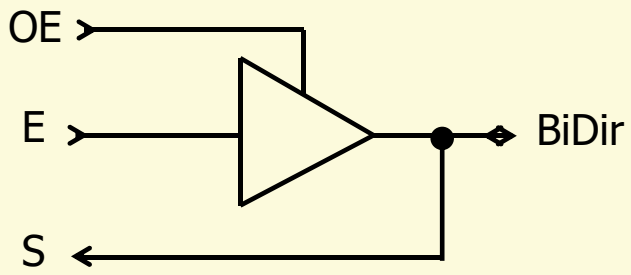
Recommandée

Porte bi-directionnelle (BiDir)

- Instanciation de porte bi-directionnelles au top
  Utiliser la fonction To_X01

```
BiDir_io <= E_i  when OE_i = '1' else 'Z';
S_o      <= To_X01(BiDir_io);
```

La fonction To_X01 permet de convertir l'état 'H' ou 'L' dans un état logique '1' ou respectivement '0'.

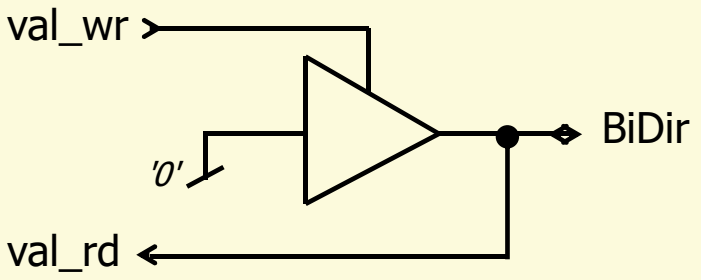


Porte collecteur ouvert (open-coll)

- Instanciation d'une porte collecteur ouvert au top
  Utiliser la fonction To_X01

```
BiDir_io <= '0'  when val_wr_s = '0' else 'Z';
val_rd_s <= To_X01(BiDir_io);
```

La fonction To_X01 permet de convertir l'état 'H' dans un état logique '1'.



Attributs prédéfinis pour les signaux …

- Ces attributs permettent de détecter des événe-ments particuliers sur les signaux :
  - permettent de détecter des flancs
  - permettent de tester les évolutions d'un signal
  - nécessaire pour la synthèse
    - uniquement détection de flanc
  - très utiles pour les test-benches et les spécifications (description avec des algorithmes)

… les attributs pour les signaux …

- Voici les principaux attributs pour les signaux :

`'event`	vrai si un événement se produit fonction rising_edge et falling_edge sont basées sur cet attribut
`'last_event`	retourne le temps écoulé depuis le dernier évènement (changement d'état)
`'last_value`	retourne la dernière valeur du signal
`'stable(T)`	vrai si le signal est resté stable durant T

Il existe d'autres attributs qui seront vu dans le cadre d'une formation avancée

… les attributs pour les signaux

- Exemple d'utilisation avec la fonction rising_edge de std_logic_1164 :

```
function rising_edge(signal S : std_ulogic)
                                    return boolean is
begin
  return(S'event and (To_X01(S) = '1')
                 and (To_X01(S'LAST_VALUE) = '0'));
end;
```

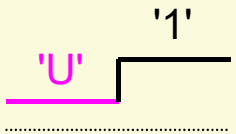
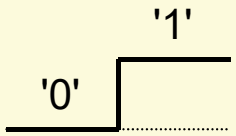
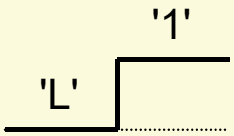
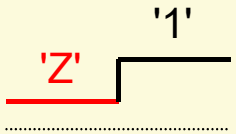
La fonction To_X01

```
function To_X01  ( s : std_ulogic ) return  X01 is
begin
  return (cvt_to_x01(s));
end;
```

Utilise la table de conversion : cvt_to_x01

```
constant cvt_to_x01 : logic_x01_table := (
                      'X',  -- 'U'
                      'X',  -- 'X'
                      '0',  -- '0'
                      '1',  -- '1'
                      'X',  -- 'Z'
                      'X',  -- 'W'
                      '0',  -- 'L'
                      '1',  -- 'H'
                      'X'   -- '-'        );
```

Exercice sur les attributs

Clk	rising_edge(Clk)	clk'event **and** clk = '1'
	.....	.....
	.....	.....
	.....	.....
	.....	.....

Nouveautés VHDL-2008

Pas supporté par Quartus Prime 18.1 !!!

Quartus 18.1 ne supporte pas les fonctionnalités suivantes :

- Exemple d'affectation de vecteurs:

  Possible avec tailles différentes (complète ou tronque)

```
vecteur_6bits  <= x"3A";          --  6 / 8
vecteur_30bits <= x"2A85_E46A"; -- 30 / 32
vecteur_12bits <= x"B8"; --complété par des '0'
```

Nouveautés VHDL-2008

Pas supporté par Quartus Prime 18.1 !!!

- Opérateurs logiques de réduction (unaire)
  - opération logique sur tous les bits d'un vecteur avec résultat sur un scalaire (bit)

```
and_all_bits <= and vect_nbits;
parity       <= xor vector;
```

- Opérateurs logiques entre scalaire et tableau
  - possible d'avoir un scalaire (bit) avec un tableau (array – vecteur)

```
en_i              : std_logic
vect_o , vect_s : std_logic_vector(7 downto 0)

vect_o <= en_i and vect_s;
```

FIN présentation VHDL !

- Questions