VHDL - Logique programmable

Partie 4
Style : description structurelle

Denis Giacona

ENSISA
École Nationale Supérieure d'Ingénieur Sud Alsace
12, rue des frères Lumière
68 093 MULHOUSE CEDEX
FRANCE

Tél. 33 (0)3 89 33 69 00

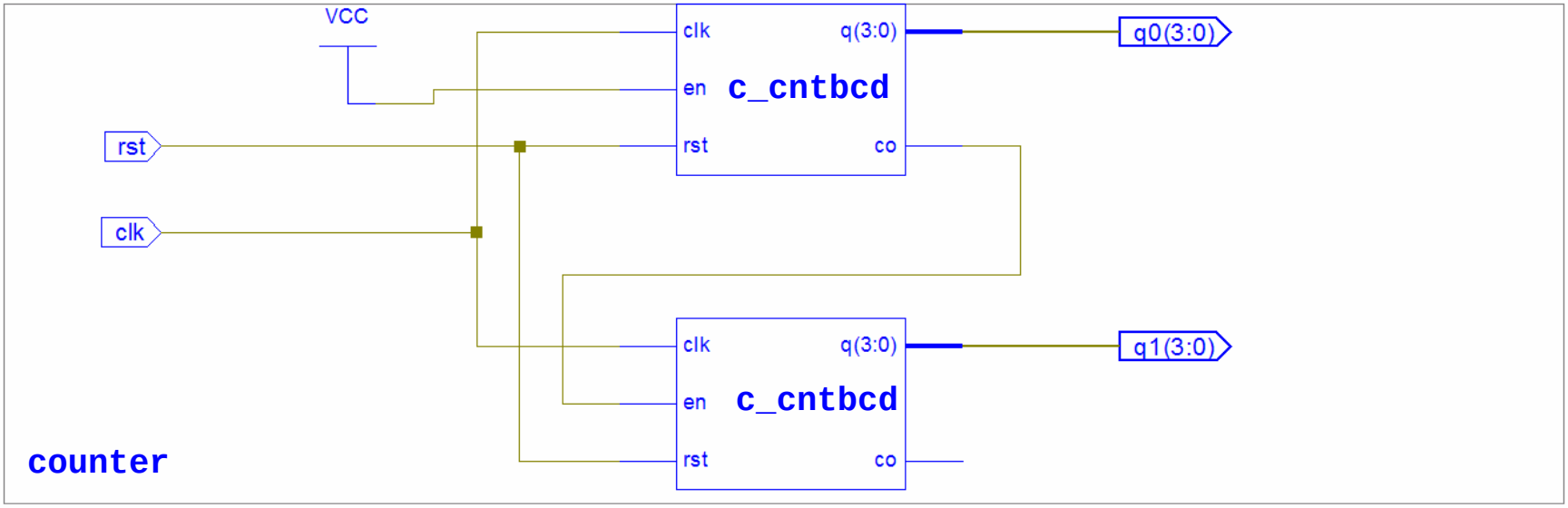
1.Du schéma à la programmation structurelle HDL

1.1. La conception HDL structurelle

1.1.1. Première phase de conception : faire une description schématique

Exemple

- **Concevoir un bloc logique `cntbcd` (compteur BCD 4 bits) et le définir comme un composant**
- **Instancier le composant deux fois pour créer un compteur BCD 2 digits `counter`**



1.1.2. Deuxième phase de conception : écrire le code VHDL

Une description structurelle VHDL est la traduction littérale d'un schéma représentant des interconnexions de blocs logiques appelés composants.

❑ Complexité globale de la fonction

- **Si faible : les composants interconnectés sont des portes élémentaires**
- **Si forte : les composants interconnectés sont des fonctions complexes définies par l'utilisateur ou fournies par un constructeur**

❑ Règles de structuration

- **Un composant peut être constitué lui-même d'autres composants**
- **Dans une architecture utilisatrice, les composants sont reliés entre eux par l'intermédiaire de signaux internes**
- **On n'affecte pas de sens (mode) à un signal interne comme on le fait pour un signal d'entité**

Remarque : Un signal interne peut disparaître au cours du processus de synthèse (sauf interdiction par une directive de compilation : `attribute synthesis_off of nom_signal : signal is true;`

1.2. Propriétés des composants

- **Un composant doit, d'une part, être déclaré, et de l'autre, être défini par un couple entité-architecture**

- **Une instance de composant est une copie distincte d'un composant dans une architecture utilisatrice**

  - **L'instanciation a pour effet d'associer des signaux effectifs (ceux de l'architecture appelante) aux signaux formels (ceux du composant)**

  - **Un composant instancié est toujours marqué d'une étiquette, à la manière d'un repère sur un schéma**

- **Un composant est soit non générique soit générique**

2.Organisation physique dans l'outil Xilinx ISE (les fichiers)

Trois possibilités :

❶ **Description dans un seul fichier, rangé dans la bibliothèque prédéfinie `work`, contenant :**
- o **l'entité/architecture du composant**
- o **l'entité/architecture utilisatrice**
  - **l'architecture utilisatrice comporte la déclaration du composant (`component`)**
  - **l'architecture utilisatrice comporte les instances du composant**

❷ **Description dans deux fichiers rangés tous deux dans la bibliothèque `work` :**
- o **un fichier contenant l'entité/architecture du composant**
- o **un fichier contenant l'entité/architecture utilisatrice**
  - **l'architecture utilisatrice comporte la déclaration du composant**
  - **l'architecture utilisatrice comporte les instances du composant**

❸ **Description dans deux fichiers (avec une bibliothèque spécifique) :**
- o **un fichier, rangé dans une bibliothèque créée par le concepteur, contenant**
  - **un paquetage incluant la déclaration de composant**
  - **et l'entité/architecture du composant**
- o **un fichier, rangé dans la bibliothèque `work`, contenant :**
  - **la déclaration d'usage de la bibliothèque (`library` et `use`)**
  - **l'entité/architecture utilisatrice**
    - **l'architecture utilisatrice comporte les instances du composant**

2.1. Description VHDL dans un seul fichier

```
-- Entité/architecture du composant
entity c_cntbcd is Port(
    …);
end c_cntbcd;

architecture Behavioral of c_cntbcd is
begin
-- Description du comportement du composant
end Behavioral;


-- Entité/architecture utilisatrice
entity counter is port(
    …);
end counter;

architecture archcounter of counter is
    -- Déclaration de composant
    COMPONENT c_cntbcd   PORT (
    …);
    END COMPONENT;
begin
    -- Description du comportement (contenant les instances du composant)
end archcounter;
```

Un seul fichier rangé dans la bibliothèque de travail **work**

2.2. Description VHDL dans deux fichiers (de la bibliothèque de travail)

```
-- Entité/architecture du composant
entity c_cntbcd is Port(
    …);
end c_cntbcd;

architecture Behavioral of c_cntbcd is
begin
-- Description du comportement du composant
end Behavioral;
```

1er fichier :
- rangé dans la bibliothèque de travail **work**

```
-- Entité/architecture utilisatrice
entity counter is port(
    …);
end counter;

architecture archcounter of counter is
    -- Déclaration de composant
    COMPONENT c_cntbcd   PORT (
    …);
    END COMPONENT;
begin
    -- Description du comportement (contenant les instances du composant)
end archcounter;
```

2e fichier :
- rangé dans la bibliothèque de travail **work**
- fichier principal « top file »

2.3. Description VHDL dans deux fichiers (avec paquetage dans une bibliothèque spécifique)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Déclaration de paquetage et de composant
PACKAGE p_cnt IS
     COMPONENT c_cntbcd  PORT (
          rst, clk, en : in std_logic;
          co : out std_logic;
          q : out std_logic_vector(3 downto 0) );
     END COMPONENT;
END p_cnt;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Entité/architecture du composant
entity c_cntbcd is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           en : in  STD_LOGIC;
           co : out  STD_LOGIC;
           q : out  STD_LOGIC_VECTOR (3 downto 0));
end c_cntbcd;

architecture Behavioral of c_cntbcd is
     -- Description du comportement du composant
```

1er fichier :
- rangé dans une bibliothèque spécifique, par exemple **lib_cnt**

☝ Même ordre des signaux

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Entité/architecture utilisatrice
entity counter is port(
     clk, rst       : in std_logic;
     q0, q1         : out std_logic_vector(3 downto 0));

end counter;

library lib_cnt;
     use lib_cnt.p_cnt.all;

architecture archcounter of counter is
     signal co0, co1     : std_logic;
     signal sq0, sq1     : std_logic_vector (3 downto 0);
     signal en0, en1     : std_logic;

begin
     en0 <= '1';
     en1 <= co0;
     bcd0: c_cntbcd port map (rst, clk, en0, co0, sq0);
     bcd1: c_cntbcd port map (rst, clk, en1, co1, sq1);
     q0 <= sq0;
     q1 <= sq1;

end archcounter;
```

2e fichier :
- rangé dans la bibliothèque de travail **work**

Déclaration d'accès aux composants qui ont été rangés dans le paquetage **p_cnt** de la bibliothèque spécifique **lib_cnt**

- Deux instanciations du composant
- Attention à l'ordre des signaux (le même que dans les déclarations d'entité)

3.Déclaration et instanciation de composants non génériques

Les déclarations des composants non génériques ne comportent que des données dont les dimensions sont fixées à l'avance.

3.1. Déclaration d'un composant non générique

```
component nom_composant
    port (
    [signal] identificateur{,identificateur}:[mode] type_signal
   {;[signal] identificateur{,identificateur}:[mode] type_signal}
    );
end component [nom_composant];
```

La déclaration est faite

- **soit dans la zone de déclaration d'une architecture (cas où le composant est décrit dans un fichier séparé sans paquetage)**
- **soit dans un paquetage visible depuis cette architecture (cas où le composant est décrit dans un fichier séparé avec paquetage et rangé dans une bibliothèque)**

3.2. Entité d'un composant non générique

```
entity nom_composant is
    port (
    [signal] identificateur{,identificateur}:[mode] type_signal
   {;[signal] identificateur{,identificateur}:[mode] type_signal}
    );
end nom_composant;
```

3.3. Instanciation d'un composant non générique, avec association des signaux par la position

```
étiquette : nom_composant
   port map (
    nom_de_signal | nom_de_variable | expression | open
   {,nom_de_signal | nom_de_variable | expression | open}
    );
```

Pour chaque instance, il convient de lister les signaux dans l'ordre défini dans la déclaration (component).

3.4. Instanciation d'un composant non générique - association des paramètres et des signaux par leur nom

- L'association des signaux peut se faire sans respecter l'ordre de la déclaration du composant ➔ le symbole **=>** est alors utilisé pour établir les correspondances entre signal formel et signal effectif
- Le symbole **=>** permet aussi de faire l'association des constantes aux paramètres génériques

3.5. Exemple de déclaration dans un paquetage d'une bibliothèque spécifique

Signaux formels

Dimensions fixes

```
package p_reg is

 component c_reg8_d port (
  clk : in std_logic;
  d   : in std_logic_vector(7 downto 0);
  q   : out std_logic_vector(7 downto 0));
 end component;

end p_reg;
```

3.6. Exemple d'instanciation : association des signaux par la position

```
library lib_cours;
use lib_cours.p_reg.all;

architecture arch_instance_non_gen_pos of instance_non_gen_pos is

signal data_int : std_logic_vector(7 downto 0);

begin

  registre1: c_reg8_d
  port map(clk,data_in,data_int);

  registre2: c_reg8_d
  port map(clk,data_int,data_out(7 downto 0));

  data_out(15 downto 8) <= data_int;

end arch_instance_non_gen_pos;
```

Référence au paquetage qui contient la déclaration du composant

Les signaux effectifs sont écrits dans le même ordre que les signaux formels (clk, d, q) dans la définition du composant

La liste des signaux figurant dans **port map** peut contenir :

- des signaux d'entrée-sortie de l'entité associée
- des signaux internes
- des constantes (☝ le passage de littéraux est interdit)
- le mot réservé **open** (lorsque l'architecture appelante n'utilise pas la sortie définie dans le composant)

4.Déclaration et instanciation de composants génériques

4.1. Intérêt d'un composant générique

Un composant générique comprend :
- **des invariants qui dénotent les caractéristiques communes à toutes ses instances**
- **des paramètres qui correspondent aux particularités de ses instances**

Exemples d'invariants
- la fonction à réaliser (compter, décaler, mémoriser, sélectionner,…), quelles que soient les dimensions des données

Exemples de paramètres
- la taille d'un registre (nombre de flip-flops)
- la taille des données d'un multiplexeur
- le nombre d'entrées de données d'un multiplexeur
- le modulo d'un compteur

Intérêts évidents de la généricité
- l'adaptabilté
- la facilité de réutilisation

4.2. Propriétés des paramètres génériques

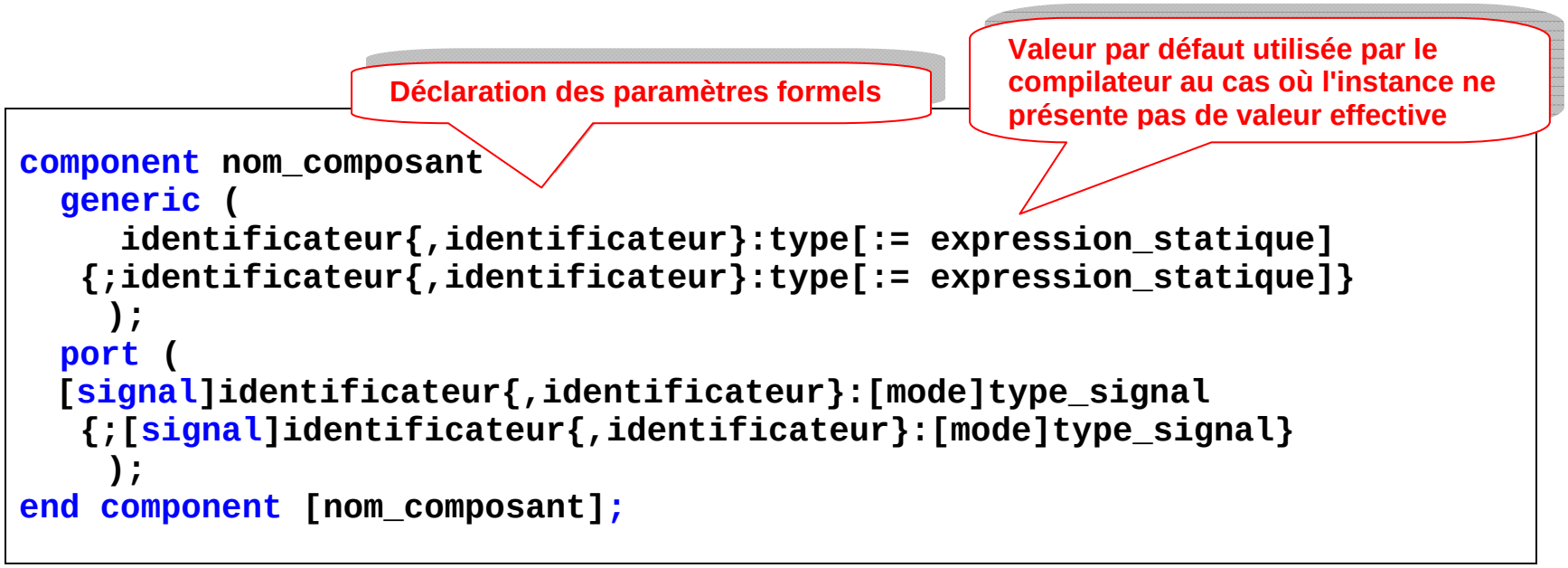
- **La déclaration des paramètres génériques formels précède celle des ports, ce qui permet aussi de paramétrer ces derniers**

- **Les paramètres génériques effectifs doivent avoir une valeur connue au moment de la compilation (avec la possibilité de définir une valeur par défaut)**

- **Dans une architecture, les paramètres génériques effectifs sont traités comme des constantes; les composants sont donc configurables au moment de leur utilisation**

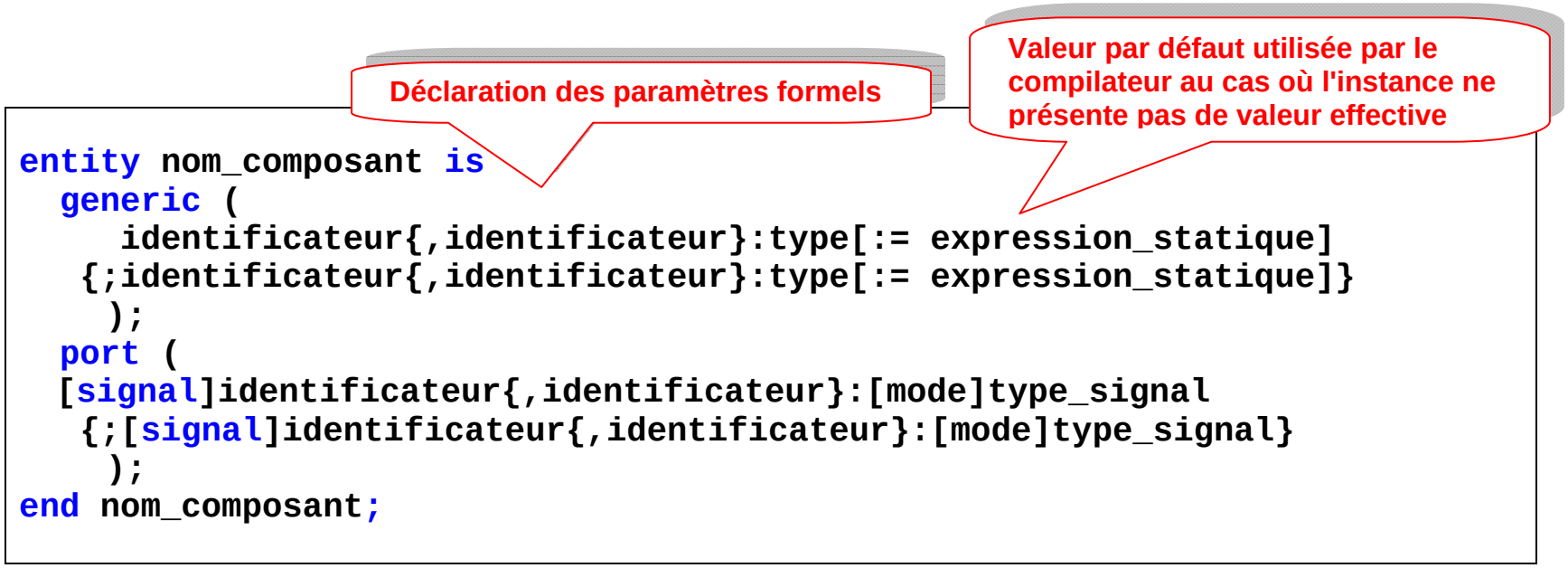
- **A l'instanciation, l'association de constantes aux paramètres peut se faire par la position ou par le nom**

4.3. Déclaration d'un composant générique



Dans la déclaration de composant, le mot clé `generic` annonce la liste des paramètres génériques formels (par ex. la taille d'un vecteur, le modulo d'un compteur)

4.4. Définition de l'entité d'un composant générique



Dans l'entité, le mot clé `generic` annonce la liste des paramètres génériques formels (par ex. la taille d'un vecteur, le modulo d'un compteur)

4.5. Instanciation d'un composant générique - association des paramètres et des signaux par la position

```
étiquette :   nom_composant
  generic map (
   expression | open {,expression | open})

  port map (
   nom_de_signal | nom_de_variable | expression | open
    {,nom_de_signal | nom_de_variable | expression | open});
```

Valeurs des paramètres effectifs

Dans l'architecture, les paramètres effectifs peuvent être utilisés comme des constantes. Ils peuvent aussi comporter des attributs (`others`, `range`, `left`, …)

4.6. Instanciation d'un composant générique - association des paramètres et des signaux par leur nom

- L'association des signaux peut se faire sans respecter l'ordre de la déclaration du composant ➔ le symbole **=>** est alors utilisé pour établir les correspondances entre signal formel et signal effectif

- Le symbole **=>** permet aussi de faire l'association des constantes aux paramètres génériques

4.7. Exemples de déclaration et de définition entité/architecture d'un composant générique dans un paquetage

Extrait d'un fichier appartenant à un projet « `library` »

```
package p_reg_d_gen is

component c_en_reg_d_gen

generic (nb_bits : natural := 4);
port(
   ar, clk, en : in std_logic;
   d           : in std_logic_vector(nb_bits - 1 downto 0);
   q           : out std_logic_vector(nb_bits - 1 downto 0));

end component;

end p_reg_d_gen;
```

Définition d'un paramètre formel dont la valeur effective par défaut est 4

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

entity c_en_reg_d_gen is
 generic (nb_bits : natural := 4);
 port(
  ar, clk, en : in std_logic;
  d           : in std_logic_vector(nb_bits - 1 downto 0);
  q           : out std_logic_vector(nb_bits - 1 downto 0));
end c_en_reg_d_gen;

architecture arch_c_en_reg_d_gen of c_en_reg_d_gen is
begin

process (ar, clk)
begin
 if ar = '1' then
  q <= (others => '0');
 elsif (clk'event and clk = '1') then
  if en = '1' then
   q <= d;
  else
   q <= q;
  end if;
 end if;
end process;

end arch_c_en_reg_d_gen;
```

4.8. Exemple d'Instanciation de composant : association des signaux par le nom

```
library lib_cours;
use lib_cours.p_reg_d_gen.all;

entity instance_gen_nom is port(
  data_in    : in std_logic_vector(7 downto 0);
  clk        : in std_logic;
  data_out   : out std_logic_vector(15 downto 0));
end instance_gen_nom;

architecture arch_instance_gen_nom of instance_gen_nom is
  signal   data_int  : std_logic_vector(data_in'range);
  constant high      : std_logic := '1';
  constant low       : std_logic := '0';
```

```
begin
-------------------------------------------------------------------
-- Assignation des signaux effectifs aux signaux formels par le nom
-------------------------------------------------------------------

registre1: c_en_reg_d_gen
generic map (data_in'length)
port map
  (ar   => low,
   clk  => clk,
   en   => high,
   d    => data_in,
   q    => data_int);


registre2: c_en_reg_d_gen
generic map (data_in'length)
port map
  (ar   => low,
   clk  => clk,
   en   => high,
   d    => data_int,
   q    => data_out(data_in'range));

data_out(15 downto 8) <= data_int;

end arch_instance_non_gen_nom;
```

Valeur du paramètre effectif (utilisation d'un attribut valeur d'un signal)

Signaux formels

Signaux effectifs

La liste des signaux figurant dans **port map** peut contenir :

- des signaux d'entrée-sortie de l'entité associée
- des signaux internes
- des constantes (le passage de littéraux est interdit)
- le mot réservé **open** (cas où l'architecture appelante n'utilise pas la sortie définie dans le composant)