System on Chip FPGA

VHDL avancé pour la synthèse

Yann Thoma

Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Février 2024

Types

type: ensemble des valeurs prises par un objet et regroupées en quatre classes:

- types scalaires (scalar): entier (integer),
  réel (real),
  énuméré (enumerated),
  physique (physical)
- types composites (composite): tableau (array),
  enregistrement (record)
- type accès (access): pointeur (pointer) permettant d'accéder à des objets d'un type donné
- type fichier (file): séquence de valeurs d'un type donné

Types énumérés: machine d'états

- Utilisation des types énumérés pour les machines d'états
- Simplifie l'écriture
- Le synthétiseur est responsable du codage
  - 1 parmi M
  - binaire
  - Gray
  - ...

Types énumérés: machine d'états

Exemple

```
type state_t is (INIT, SEND, WAITING);

signal state, next_state : state_t;

process(all) is
begin
    next_state <= state;
    case state is
    when INIT =>
        if (...) then
            next_state <= SEND;
        end if;
    when SEND =>
        ...
    when WAITING =>
        ...
    end case;
end process;
```

Attributs d'un tableau

- Les tableaux possèdent des attributs
- Les attributs d'un tableau T s'utilisent de la façon suivante:
  - `T'nom_attribut(numero_dimension)`
- Le numéro de la dimension peut être omis et vaut dans ce cas 1.
- Les attributs suivants sont définis sur n'importe quel type tableau:
  - `LEFT`: élément le plus à gauche de l'intervalle de l'index
  - `RIGHT`: élément le plus à droite de l'intervalle de l'index
  - `HIGH`: élément le plus grand de l'index
  - `LOW`: élément le plus petit de l'index
  - `RANGE`: sous-type des indices, intervalle entre l'attribut `LEFT` et `RIGHT`
  - `REVERSE_RANGE`: intervalle inverse de `RANGE`
  - `LENGTH`: nombre d'éléments du tableau

Attributs d'un tableau: exemple

Soit le code suivant:

```
type index1 is range 1 to 20;
type index2 is range 19 downto 2;
type vecteur1 is index1 of std_logic;
type vecteur2 is index2 of std_logic;
```

Déterminer les valeurs des attributs suivants:

Attribut	vecteur1	vecteur2
LEFT		
RIGHT		
HIGH		
LOW		
RANGE		
REVERSE_RANGE		
LENGTH		

Attributs d'un tableau: usage

- Permet de rendre le code générique
- Pas besoin de connaître à l'avance la taille d'un tableau

Exemple

```
entity mux2 is
port (
    data0_i   : in  std_logic_vector;
    data1_i   : in  std_logic_vector;
    sel_i     : in  std_logic;
    dataout_o : out std_logic_vector
);

architecture behave of mux2 is
begin
    dataout_o <= data0_i when sel_i = '0' else data1_i;
end behave;
```

Exemple 2

```
entity alu is
port (
    data0_i  : in  std_logic_vector;
    data1_i  : in  std_logic_vector;
    mode_i   : in  std_logic_vector(1 downto 0);
    result_o : out std_logic_vector;
    carry_o  : out std_logic;
);
architecture behave of alu is
    signal d0_s : std_logic_vector(data0_i'high + 1 downto 0);
    signal d1_s : std_logic_vector(data0_i'high + 1 downto 0);
    signal r_s  : std_logic_vector(data0_i'high + 1 downto 0);
begin
    d0_s <= "0" & data0_i;
    d1_s <= "0" & data1_i;
    r_s <= unsigned(d0_s) + unsigned(d1_s) when mode_i = "00" else
           unsigned(d0_s) - unsigned(d1_s) when mode_i = "01" else ...;
    result_o <= r_s(result_o'range);
    carry_o <= r_s(r_s'high);
end behave;
```

Instanciation

- Les tailles des vecteurs non contraints ne sont pas définies à la compilation
- Elles sont fixées à l'élaboration
- Une bonne idée:
  - Ajouter des assertions sur les tailles

Exemple de l'ALU

```
architecture behave of alu is
    signal d0_s : std_logic_vector(data0_i'high + 1 downto 0);
    signal d1_s : std_logic_vector(data0_i'high + 1 downto 0);
    signal r_s  : std_logic_vector(data0_i'high + 1 downto 0);
begin
    assert data0_i'range = data1_i'range;
    assert data0_i'range = result_o'range;
    d0_s <= "0" & data0_i;
    d1_s <= "0" & data1_i;
    r_s <= unsigned(d0_s) + unsigned(d1_s) when mode_i = "00" else
           unsigned(d0_s) - unsigned(d1_s) when mode_i = "01" else ...;
    result_o <= r_s(result_o'range);
    carry_o <= r_s(r_s'high);
end behave;
```

Types composites: enregistrements

Collection d'éléments nommés, ou champs, dont les valeurs peuvent être de types différents

Exemple

```
type memory_bus_in_t is record
    addr   : std_logic_vector(15 downto 0);
    data   : std_logic_vector(7 downto 0);
    read   : std_logic;
    write  : std_logic;
end record memory_bus_in_t;
```

Accès aux éléments de l'enregistrement

```
signal MB: memory_bus_in_t;
MB.addr                 -- tout le tableau addr
MB.addr(7 downto 0)     -- une partie du tableau addr
MB.data(7)              -- bit de donnée de poids fort
```

Types composites: enregistrements

- Les enregistrements permettent de représenter des bus
- A utiliser pour les ports des entités
- Simplifie la connectivité

Exemple

```
entity memory is
port (
    bus_in_i  : memory_bus_in_t;
    bus_out_o : memory_bus_out_t
);
end memory;
```

Le processus

Un processus:

- doit être déclaré dans le corps de l'architecture.
- regroupe du code dont l'exécution est séquentielle.
- s'exécute en un temps Δ, il y a donc un "avant" l'exécution et un "après" l'exécution. Le temps Δ est toutefois infinitésimal.
- est considéré comme une instruction concurrente.
- a une durée de vie éternelle. (quelle chance...)
- ne peut être créé dynamiquement.
- peut accéder à tous les signaux déclarés dans l'entité et l'architecture.

Processus: liste de sensibilité

- La liste de sensibilité d'un processus énumère l'ensemble des signaux qui, lorsqu'ils changent de valeurs, entraînent le réveil du processus et son exécution.
- Une affectation concurrente peut être considérée comme un processus ayant comme liste de sensibilité l'entièreté des signaux utilisés en lecture.

concurrent

```
a <= b and c;
```

$\cong$

séquentiel

```
process(all)
begin
    a <= b and c;
end process;
```

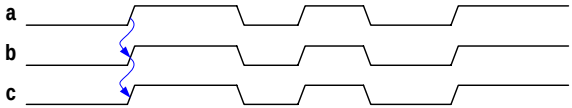
Comportement des signaux

- VHDL concurrent:

```
b <= a;
c <= b;
```

≃

```
b <= a;
c <= a;
```



- VHDL séquentiel équivalent:

```
process(all)
begin
    b <= a;
    c <= b;
end process;
```

Attention mauvaise pratique

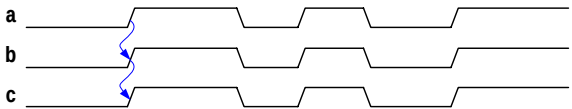
Comportement des signaux: mauvais exemple

- VHDL séquentiel:

```
process(a,b)
begin
    b <= a;
    c <= b;
end process;
```

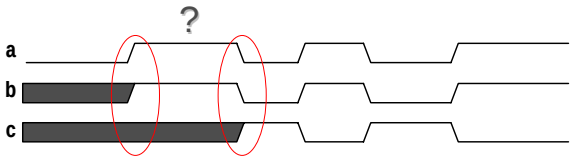
$\simeq$

```
c <= a;
```



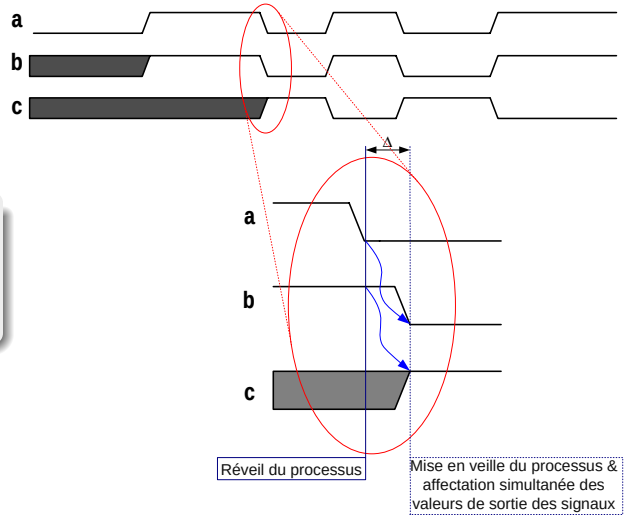
- VHDL séquentiel (mauvais exemple):

```
process(a)
begin
    b <= a;
    c <= b;
end process;
```



Comportement des signaux: mauvais exemple détaillé

```
process(a)
begin
    b <= a;
    c <= b;
end process;
```



Comportement des signaux: bonne pratique

- Dans un processus, ne pas utiliser les signaux affectés

Pas bon

```
process(a , b)
begin
    b <= a;
    c <= b;
end process;
```

Bon

```
process(a)
begin
    b <= a;
    c <= a;
end process;
```

Meilleur

```
process(all)
begin
    b <= a;
    c <= a;
end process;
```

VHDL: processus combinatoires, règles à respecter

- La liste de sensibilité d'un processus combinatoire doit contenir tous les signaux source (utilisés à droite d'une affectation ou dans un test)
- Exemples
  - `a <= b and c;`
  - `if q='1' then ...`
  - `case state is ...`

Bonne pratique

- Utiliser `all` pour les processus combinatoires
- Utiliser le reset asynchrone et l'horloge pour les processus séquentiels

Processus combinatoire

```
process(all) is
begin
    if (a = '1') then
        c <= b;
    end if;
end process;
```

Processus séquentiel

```
process(reset, clk) is
begin
    if (reset = '1' ) then
        a <= '0';
    elsif rising_edge(clk) then
        a <= b;
    end if;
end process;
```

VHDL: processus combinatoires, règles à respecter

- Affecter des valeurs par défaut à tous les signaux affectés dans le processus
  - Evite la génération de latchs

Exemple (génère un latch)

```
process(all) is
begin
    if (e = '1') then
        a <= b;
    end if;
end process;
```

Exemple (bon)

```
process(all) is
begin
    a <= '0';
    if (e = '1') then
        a <= b;
    end if;
end process;
```

If

Syntaxe

```
[label_if]:if <expression booléenne> then
    ...
elsif <expression booléenne> then
    ...
else
    ...
end if [label_if];
```

Exemple

```
if sel = "00" then
    output <= a;
elsif sel = "01" then
    output <= b;
else
    output <= c;
end if;
```

Case

Syntaxe

```
[label_case]:case <expression> is
    when <choix> => ...;
    when <choix> => ...;
    when others  => ...;
end case [label_case];
```

Il est possible pour `choix` de mettre plusieurs valeurs ou un espace, par exemple:

Syntaxe

```
    when add|sub|load   => ...
    -- ou
    when 0 to 15        => ...
```

Case: exemples

Exemple 1

```
type state_t is (Init,Fetch,Writeback,Done);
variable state : state_t;

case state is
    when Init            => ...;
    when Fetch|Writeback => ...;
    when others          => ...;
end case;
```

Exemple 2

```
variable a : integer range 0 to 15;

case a is
    when 0       => ...;
    when 1|13    => ...;
    when 2 to 10 => ...;
    when others  => ...;
end case;
```

Loop

Syntaxe

```
[label_loop]:loop
    ...
end loop [label_loop];
```

Sortie de boucle

```
exit [label_loop];
exit [label_loop] when <expression booléenne>;
```

Instruction next

```
next [label_loop];
next [label_loop] when  <expression booléenne>;
```

Loop: exemple

Exemple 1

```
loop
    ...
    if a=10 then
        next;
    end if;
    ...
    if a > 15 then
        exit;
    end if;
end loop;
```

Exemple 2

```
loop
    ...
    next when a=10;
    ...
    exit when a > 15;
end loop;
```

Exemple 3

```
aloop:loop
    anotherloop:loop
        ...
        next aloop when a=10;
        ...
        exit when a > 15;
-- quitte la boucle anotherloop
    end loop
end loop;
```

While

Syntaxe

```
[label_loop]:while <condition> loop
    ...
end loop [label_loop];
```

Exemple

```
while a>15 loop
    ...
end loop;
```

For

Syntaxe

```
[label_loop]:for <identificateur> in <intervalle> loop
    ...
end loop [label_loop];
```

Exemple

```
for i in 0 to 7 loop
    ...
end loop;

type semaine is (lun,mar,mer,jeu,ven,sam,dim);
...
for jours in semaine'RANGE loop
    ...
end loop;
```

Usage des boucles

- Attention à l'usage des boucles
- Mène facilement à un mauvais résultat de synthèse
- Il faut s'imaginer ce que la boucle va générer

Les variables

- Les variables sont déclarées dans les processus. Elles n'ont de portée que dans le processus de déclaration.
- Les variables gardent leurs valeurs entre deux réveils d'un processus.
- Les variables changent de valeurs immédiatement suite à leur affectation, contrairement aux signaux.
- Trois règles à appliquer concernant les variables:
  - Une variable qui, après synthèse, a été substituée par un signal ou un registre est une mauvaise variable.
  - Les variables sont faites pour simplifier l'expression de problèmes spatialement itératifs. Une variable ne devrait pas être utilisée temporellement. Un générateur de parité peut facilement s'exprimer à l'aide d'une variable, mais utiliser une variable pour réaliser un registre à décalage, bien que correct, est à prohiber (selon la règle précédente).
  - Une variable ne doit pas avoir de valeur par défaut à la déclaration

Utilisation des variables : bonne pratique

```
process(all) is
    variable var_v : std_logic;
begin
    var_v := default_value;          <-- Donner une valeur par défaut
    ...

    var_v := some_function(some_signal);   <-- Modifications de la variable
    ...

    another_signal <= var_v;         <-- Affectation en fin de process
end process;
```

Utilisation des variables

Exemple: And8

```
entity and_8 is
    port (a_i : in  std_logic_vector(7 downto 0);
          z_o : out std_logic);
end and_8;
architecture behave of and_8 is
begin
   process(all)
   variable calcul_v : std_logic;
   begin
      calcul_v := '1';
      for i in a_i'range loop
         if a_i(i) = '0' then
            calcul_v := '0';
            exit;
         end if;
      end loop;
      z_o <= calcul_v;
   end process;
end behave;
```

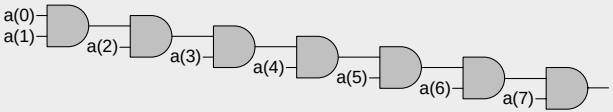
Code générique

Utilisation des variables

Exemple: And8

```
entity and_8 is
    port (a_i : in  std_logic_vector(7 downto 0);
          z_o : out std_logic);
end and_8;

architecture behave2 of and_8 is
begin
   process(all)
   variable calcul_v : std_logic;
   begin
      calcul_v := '1';
      for i in a_i'range loop
         calcul_v := calcul_v and a_i(i);
      end loop;
      z_o <= calcul_v;
   end process;
end behave2;
```



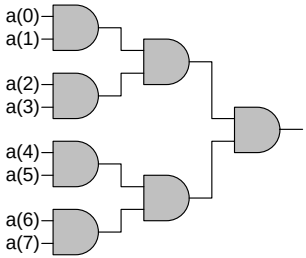
Utilisation des variables

Exemple: And8

```
entity and_8 is
    port (a_i : in std_logic_vector(7 downto 0);
          z_o : out std_logic);
end and_8;

architecture behave2 of and_8 is
begin
   process(all)
   variable calcul_v : std_logic;
   begin
      calcul_v := '1';
      for i in a_i'range loop
         calcul_v := calcul_v and a_i(i);
      end loop;
      z_o <= calcul_v;
   end process;
end behave2;
```

Solution optimisée:

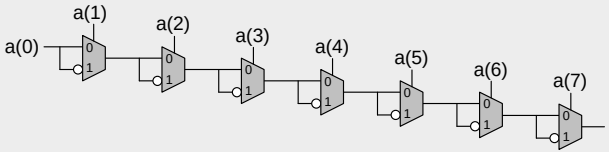


Utilisation des variables

Exemple: détecteur de parité

```
entity parity is
    port (a_i : in  std_logic_vector(7 downto 0);
          z_o : out std_logic);
end parite;

architecture behave of parity is
begin
   process(all)
   variable calcul_v : std_logic;
   begin
      calcul_v := '0';
      for i in a_i'range loop
         if a_i(i) = '1' then
            calcul_v := not calcul_v;
         end if;
      end loop;
      z_o <= calcul_v;
   end process;
end behave;
```

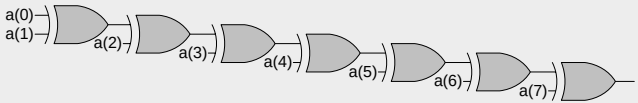


Utilisation des variables

Exemple: détecteur de parité

```
entity parity is
    port (a_i : in  std_logic_vector(7 downto 0);
          z_o : out std_logic);
end parity;

architecture behave of parity is
begin
   process(all)
   variable calcul_v : std_logic;
   begin
      calcul_v := '0';
      for i in a_i'range loop
         if a_i(i) = '1' then
            calcul_v := not calcul_v;
         end if;
      end loop;
      z_o <= calcul_v;
   end process;
end behave;
```

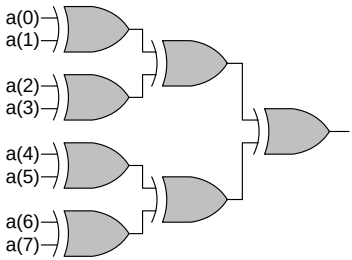


Utilisation des variables

Exemple: détecteur de parité

```
entity parity is
    port (a_i : in  std_logic_vector(7 downto 0);
          z_o : out std_logic);
end parity;
architecture behave of parity is
begin
   process(all)
   variable calcul_v : std_logic;
   begin
      calcul_v := '0';
      for i in a_i'range loop
         if a_i(i) = '1' then
            calcul_v := not calcul_v;
         end if;
      end loop;
      z_o <= calcul_v;
   end process;
end behave;
```

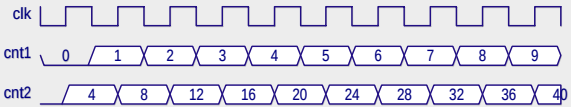
Solution optimisée:



Comportement des signaux et variables (1)

Attention code pas top

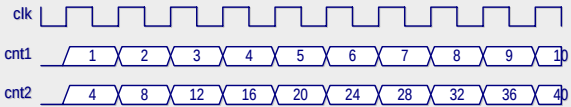
```
entity test is
port ( clk : in std_logic;
       cnt1 : out unsigned (7 downto 0);
       cnt2 : out unsigned (7 downto 0));
end test;
architecture behave1 of test is
signal c1 : unsigned (7 downto 0) := (others => '0');
begin
    process(clk)
    variable c2 : unsigned (7 downto 0) := (others => '0');
    begin
       if rising_edge(clk) then
          for i in 0 to 3 loop
             c1 <= c1 + 1;
             c2 := c2 + 1;
          end loop;
       end if;
       cnt1 <= c1;
       cnt2 <= c2;
    end process;
```



Comportement des signaux et variables (2)

Attention code pas top

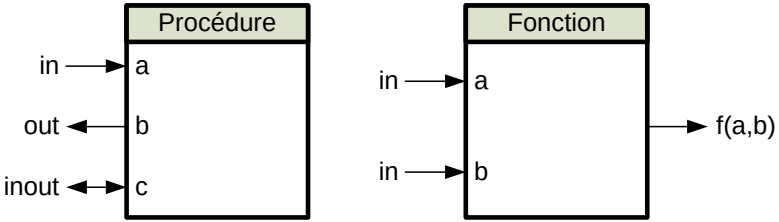
```
entity test is
port ( clk : in std_logic;
       cnt1 : out unsigned (7 downto 0);
       cnt2 : out unsigned (7 downto 0));
end test;
architecture behave2 of test is
signal c1 : unsigned (7 downto 0) := (others => '0');
begin
    process(clk)
    variable c2 : unsigned (7 downto 0) := (others => '0');
    begin
       if rising_edge(clk) then
          for i in 0 to 3 loop
             c1 <= c1 + 1;
             c2 := c2 + 1;
          end loop;
       end if;
       cnt2 <= c2;
    end process;
    cnt1 <= c1;
```



Programmation modulaire

- "Logicielle"
  - Sous-programmes: fonctions, procédures
  - Paquetages
  - Librairies
- "Matérielle"
  - Construction hiérarchique

Sous-programmes



Procédures

Syntaxe

```
procedure nom[(liste_des_parametres)] is
    zone_declarative
begin
    zone d'instructions séquentielles
end [procedure] [nom];
```

Exemple

```
procedure min(a,b: in integer; c: out integer) is
begin
    if a < b then
        c := a;
    else
        c := b;
    end if;
end min;
```

Fonctions

Syntaxe

```
function nom[(liste_des_parametres)] return type is
    zone_declarative
begin
    zone d'instructions séquentielles
end [function] [nom];
```

Exemple

```
function min(a,b: in integer) return integer is
begin
    if a < b then
        return a;
    else
        return b;
    end if;
end min;
```

Appel de sous-programmes

- L'appel d'un sous-programme peut être concurrent ou séquentiel!

Exemple

```
-- procédure
min(var1, 5, resultat);           -- appel par position des paramètres
min(a=>var1, b=>5, c=>resultat); -- appel par nom des paramètres
min(b=>5, c=>resultat, a=>var1);

-- fonction
resultat := min(var1,5);         -- appel par position des paramètres
resultat := min(a=>var1, b=>5); -- appel par nom des paramètres
```

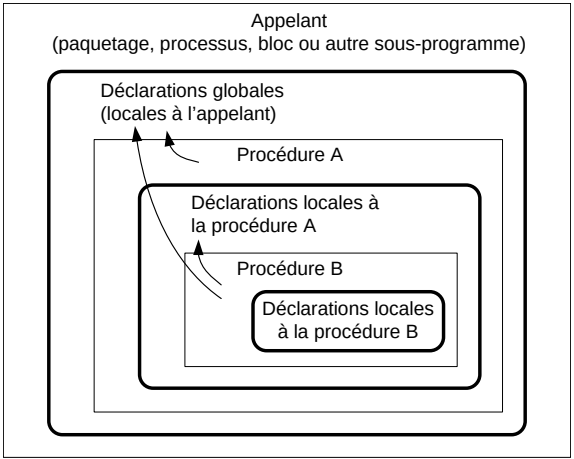
Exemple: Fonction de résolution

- Fonction de résolution du type r_bit, qui comprend les valeurs '0', '1', 'Z', et 'X'

Exemple

```
function r_resolution(sources: r_bit_vector) return r_bit is
   variable resultat: r_bit := 'Z';
begin
    for i in sources'range loop
        case sources(i) is
            when 'X' => return 'X';
            when '0' => if resultat='1' then
                            return 'X'; else resultat := '0';
                        end if;
            when '1' => if resultat='0' then
                            return 'X'; else resultat := '1';
                        end if;
            when 'Z' => null;
        end case;
    end loop;
    return resultat;
end r_resolution;
```

Sous-programmes: visibilité



Généricité

- La généricité permet de transmettre une information statique à un bloc.
- Un bloc générique est vu de l'extérieur comme un bloc paramétré.
- A l'intérieur du bloc, les paramètres génériques sont identiques à des constantes.
- Une entité peut être générique, mais pas une architecture.

Syntaxe

```
generic (param1 [, autre_param]: type_param [:=valeur_par_defaut];
         param2 [, autre_param]: type_param [:=valeur_par_defaut];
         ...
         paramN [, autre_param]: type_param [:=valeur_par_defaut]);
```

Généricité: exemple

- Une porte ET à N entrées

Porte ET

```
entity AND_gate is
    generic ( NB_INPUT : natural := 2);
    port ( input_i   : in  std_logic_vector(NB_INPUT-1 downto 0);
           output_o  : out std_logic);
end AND_gate;

architecture behave of AND_gate is
begin
    process(all)
        variable result_v: std_logic;
    begin
        result_v := '1';
        for i in 0 to NB_INPUT-1 loop
            result_v := result_v and input_i(i);
        end loop;
        output_o <= result_v;
    end process;
end behave;
```

Généricité: exemple instancié

Instanciation de la Porte ET

```
architecture struct of something is

component AND_gate is
    generic ( NB_INPUT : natural := 2);
    port ( input_i   : in  std_logic_vector(NB_INPUT-1 downto 0);
           output_o  : out std_logic);
end component;

signal the_inputs_s : std_logic_vector(7 downto 0);
signal the_output_s : std_logic;
...
begin

the_gate: AND_gate
generic map ( NB_INPUT => 8)
port map ( input_i  => the_inputs_s,
           output_o => the_output_s);
...
end struct;
```

Généricité: exemple 2

Multiplexeur de N bits

```
entity mux is
generic ( SIZE: positive := 1);
port ( input0_i : in  std_logic_vector(SIZE-1 downto 0);
       input1_i : in  std_logic_vector(SIZE-1 downto 0);
       sel_i    : in  std_logic;
       output_o : out std_logic_vector(SIZE-1 downto 0));
end mux;

architecture dataflow of mux is
begin
    output_o <= input0_i when sel_i = '0' else
                input1_i;
end flot;
```

Généricité: exemple 2

Multiplexeur de N bits

```
entity mux is
generic ( SIZE: positive := 1);
port ( input0_i : in  std_logic_vector(SIZE-1 downto 0);
       input1_i : in  std_logic_vector(SIZE-1 downto 0);
       sel_i    : in  std_logic;
       output_o : out std_logic_vector(SIZE-1 downto 0));
end mux;

architecture comp of mux is
begin
    process(all)
    begin
        for i in 0 to SIZE-1 loop
          output_o(i) <= (input0_i(i) and (not sel_i)) or
                         (input1_i(i) and sel_i);
        end loop;
    end process;
end comp;
```

Exemples de généricité

- Une mémoire: nombre de mots + taille des mots
- FIFO : nombre de mots + taille des mots
- Calculateur : taille des opérandes
- Compteur : nombre de bits

Argument pour la généricité

- Même si la généricité n'est pas nécessaire...
- Elle peut permettre d'optimiser la vérification
- Simulations plus rapides si la taille des données est plus faible
- Vérification formelle plus abordable également

Autre généricité: Tableaux non contraints

- En VHDL-2008 il est possible d'utiliser des tableaux non contraints

Multiplexeur de N bits

```
entity mux is
port ( input0_i : in  std_logic_vector;   <-- Pas de paramètre générique
       input1_i : in  std_logic_vector;
       sel_i    : in  std_logic;
       output_o : out std_logic_vector);
end mux;
architecture comp of mux is
begin
    assert(input0_i'length = input1_i'length);   <-- Idem pour output
    process(all)
    begin
        for i in input0_i'range loop
          output_o(i) <= (input0_i(i) and (not sel_i)) or
                         (input1_i(i) and sel_i);
        end loop;
    end process;
end comp;
```

Autre généricité: Tableaux non contraints

Instanciation du multiplexeur de N bits

```
signal a_s   : std_logic_vector(7 downto 0);
signal b_s   : std_logic_vector(7 downto 0);
signal sel_s : std_logic;
signal r_s   : std_logic_vector(7 downto 0);

my_mux : mux
port map(
    input0_i => a_s,
    input1_i => b_s,
    sel_i    => sel_s,
    output_o => r_s
);
```

Generate

Syntaxe (forme conditionnelle)

```
label: if condition_booléenne generate
    ...
    ... Suite d'instructions concurrentes
    ...
end generate [label];
```

VHDL-2008: Extension de la syntaxe

```
label: if condition_booléenne generate
    ...
    ... Suite d'instructions concurrentes
    ...
elsif autre_condition generate
    ...
    ... Suite d'instructions concurrentes
    ...
end generate [label];
```

Generate

Syntaxe (forme conditionnelle)

```
label: case sel
when 0 => generate
    ...
    ... Suite d'instructions concurrentes
    ...
when 1 => generate
    ...
    ... Suite d'instructions concurrentes
    ...
when others =>
    ...
    ... Suite d'instructions concurrentes
    ...
end generate [label];
```

Generate

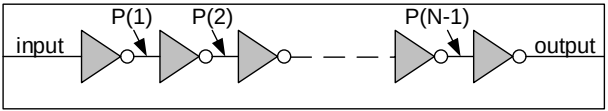
Syntaxe (forme itérative)

```
label: for nom_parametre in intervalle generate
    ...
    ... Suite d'instructions concurrentes
    ...
end generate [label];
```

Generate: exemple

Une chaîne d'inverseurs

```
entity inverters_chain is
    generic (N: integer);
    port(input_i  : in  std_logic;
         output_o : out std_logic);
end inverters_chain;
```



Generate: exemple

```
architecture struct of inverters_chain is

signal p_s : std_logic_vector(N downto 0);

component inverter is
    port(e_i: in std_logic; s_o: out std_logic);
end component;

begin
    the_inverters: for i in 0 to N-1 generate
        inv: inverter port map (p_s(i),p_s(i+1));
    end generate the_inverters;

    p_s(0)   <= input_i;
    output_o <= p_s(N);
end struct;
```

Paquetages

- Un paquetage sert à partager du code général à un projet.
- La spécification d'un paquetage présente tout ce qu'exporte le paquetage:
  - constantes
  - fichiers
  - types, sous-types
  - sous-programmes
  - déclarations de composants
  - clauses `use`
  - ...
- Le corps du paquetage contient:
  - Les sous-programmes exportés
  - Des déclarations locales (types, constantes, ...)
  - Pas de déclarations de signaux!!!
- La référence à un paquetage se fait par la clause `use`.

Exemple: spécification de paquetage

Exemple: spécification de paquetage

```
package example_pkg is

    type state_t is (init, read, write, done);

    constant MEMSIZE      : integer   := 256;
    constant RESET_ACTIVE : std_logic := '0';

    type memory_t is array(0 to MEMSIZE-1) of
                        std_logic_vector(15 downto 0);

    component my_component is
    port (
        a_i,b_i : in  std_logic;
        c_o     : out std_logic);
    end my_component;

    function min(a,b: in integer) return integer;
    function max(a,b: in integer) return integer;
end example_pkg;
```

Exemple: corps de paquetage

Exemple: corps de paquetage

```
package body example_pkg is
    function min(a,b: in integer) return integer is
    begin
        if a < b then return a;
        else return b;
        end if;
    end min;

    function max(a,b: in integer) return integer is
    begin
        if a > b then return a;
        else return b;
        end if;
    end max;
end example_pkg;
```

Utilisation de paquetages

- Pour l'utilisation d'un paquetage il faut le déclarer en début de fichier:

Exemple

```
use work.example_pkg.all;
```
← Si le paquetage est compilé dans work