Conception de Systèmes Numériques sur FPGA

Introduction

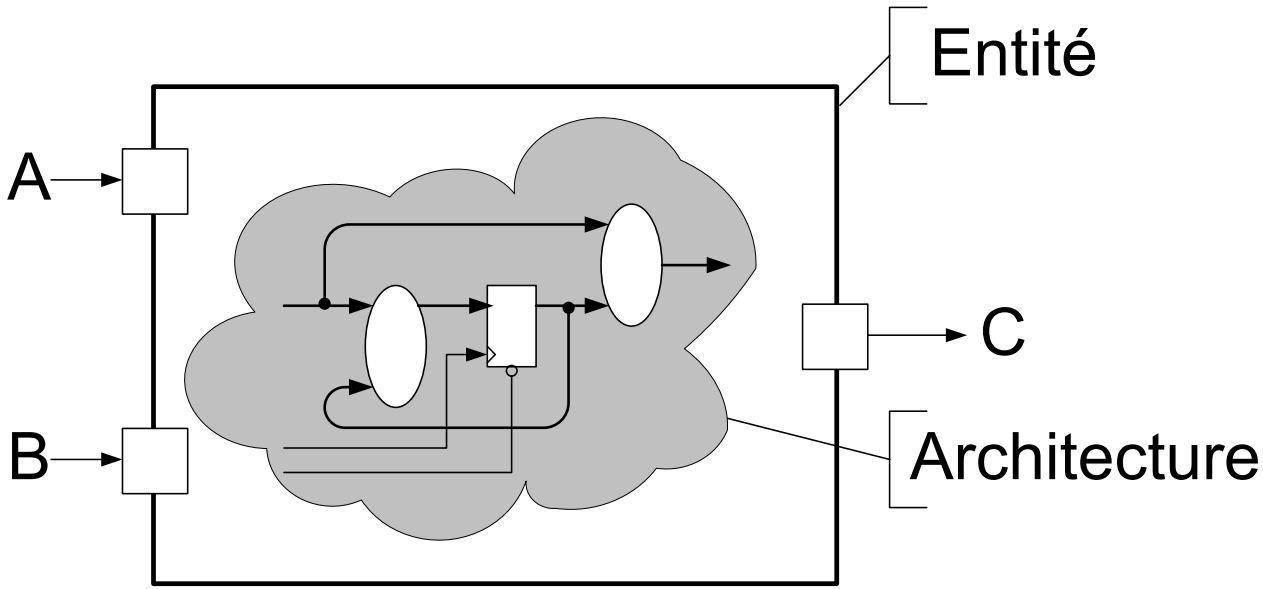
Yann Thoma

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

Février 2017

Entité et architecture

- Une *entité* peut être considérée comme une boîte noire, dont l'interface est défini et dont le contenu est invisible.
- Une *architecture* décrit le contenu de cette boîte noire. Nous pouvons déjà noter que plusieurs architectures peuvent être associées à une entité.



Entité de conception: exemple

Multiplexeur particulier

```
entity mux_inv is
    port (
        a_i    : in     std_logic;
        b_i    : in     std_logic;
        sel_i  : in     std_logic;
        c_o    : out    std_logic;
        cinv_o: out    std_logic
    );
end mux_inv;

architecture dataflow of
mux_inv is
signal out_int_s: std_logic;
begin
  out_int_s <= a_i when sel_i='0'
               else b_i;
  c_o          <= out_int_s;
  cinv_o       <= not out_int_s;
end dataflow;
```

Suite...

```
architecture behave of mux_inv is
begin
    process(a_i,b_i,sel_i)
    variable out_int_v: std_logic;
    begin
        if sel_i='0' then
            out_int_v := a_i;
        else
            out_int_v := b_i;
        end if;
        c_o     <= out_int_v;
        cinv_o <= not out_int_v;
    end process;
end behave;
```

Unités de conception

Il existe cinq types d'unités de conception:

- Déclaration d'entité (P)
- Corps d'architecture (S)
- Déclaration de paquetage (P)
- Corps de paquetage (S)
- Déclaration de configuration (P)

Une unité primaire (P) doit doit être analysée (compilée) avant son unité secondaire (S) correspondante, et toute déclaration faite dans une unité primaire est visible dans toute unité secondaire correspondante.

Entité de conception

Déclaration d'entité

```
{clause_contexte} entity nom-entité is
    [generic (liste-paramètres);]
    [port (liste-ports);]
    [déclarations-locales]
[begin
    {instruction-concurrente-passive}]
end [entity] [nom-entité];
```

Corps d'architecture

```
architecture nom-arch of nom-entité is
    [déclarations-locales]
begin
    {instruction-concurrente}}
end [architecture] [nom-arch];
```

Entité de conception

Déclaration d'entité

```
{clause_contexte} entity nom-entité is
    [generic (liste-paramètres);]
    [port (liste-ports);]
    [déclarations-locales: type, sous-type
                           constante
                           signal
                           sous-programme]
[begin
    {instruction-concurrente-passive}]
end [entity] [nom-entité;
```

Corps d'architecture

```
architecture nom-arch of nom-entité is
    [déclarations-locales]
begin
    {instruction-concurrente}
end [architecture] [nom-arch];
```

Entité de conception

Déclaration d'entité

```
{clause_contexte} entity nom-entité is
    [generic (liste-paramètres);]
    [port (liste-ports);]
    [déclarations-locales]
[begin
    {instruction-concurrente-passive: appel concurrent de procédure
                                      assertion
                                      processus passif}]
end [entity] [nom-entité];
```

Corps d'architecture

```
architecture nom-arch of nom-entité is
    [déclarations-locales]
begin
    {instruction-concurrente}
end [architecture] [nom-arch];
```

Entité de conception

Déclaration d'entité

```
{clause_contexte} entity nom-entité is
    [generic (liste-paramètres);]
    [port (liste-ports);]
    [déclarations-locales]
[begin
    {instruction-concurrente-passive}]
end [entity] [nom-entité];
```

Corps d'architecture

```
architecture nom-arch of nom-entité is
    [déclarations-locales: type, sous-type
                           constante, fichier
                           signal
                           sous-programme
                           déclaration de composant]
begin
    {instruction-concurrente}
end [architecture] [nom-arch];
```

Entité de conception

Déclaration d'entité

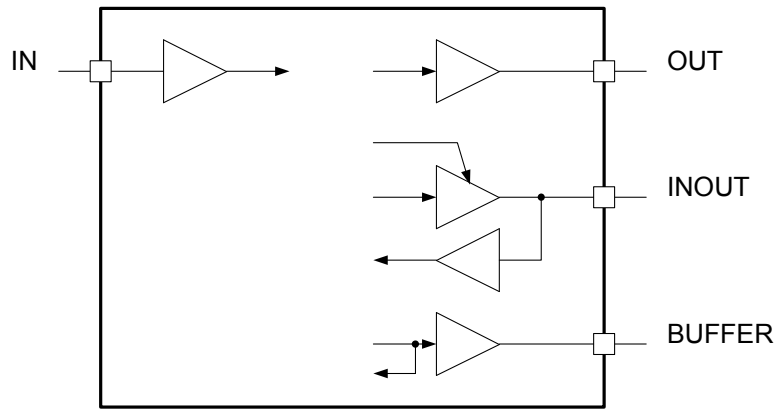
```
{clause_contexte} entity nom-entité is
    [generic (liste-paramètres);]
    [port (liste-ports);]
    [déclarations-locales]
[begin
    {instruction-concurrente-passive}]
end [entity] [nom-entité];
```

Corps d'architecture

```
architecture nom-arch of nom-entité is
    [déclarations-locales]
begin
    {instruction-concurrente: affectation concurrente de signal
                              processus
                              appel concurrent de procédure
                              assertion
                              instance de composant
                              instruction generate}
end [architecture] [nom-arch];
```

Les modes d'entrée/sortie

- Un port déclaré en mode `IN` ne peut être que lu.
- Un port déclaré en mode `OUT` ne peut être qu'écrit.
- Un port déclaré en mode `INOUT` peut être lu et écrit. Il doit par contre absolument recevoir une valeur du monde extérieur.
- Un port déclaré en mode `BUFFER` peut être lu et écrit. Il ne peut pas recevoir de valeur du monde extérieur.



Les modes d'entrée/sortie: suggestion

- Il sera préférable de ne pas utiliser le mode `BUFFER`, mais de le remplacer par le mode `OUT`
- Utilisation de signaux internes comme suit:

```
entity exemple is
port ( clk: in std_logic; a_i, b_i: in std_logic; c_o: out std_logic);
end exemple;

architecture behave of exemple is
signal c_int: std_logic;
begin
    c_o <= c_int;
    process(clk)
    begin
        if rising_edge(clk) then
            if a_i = b_i then
                c_int <= not c_int;
            end if;
        end if;
    end process;
end behave;
```

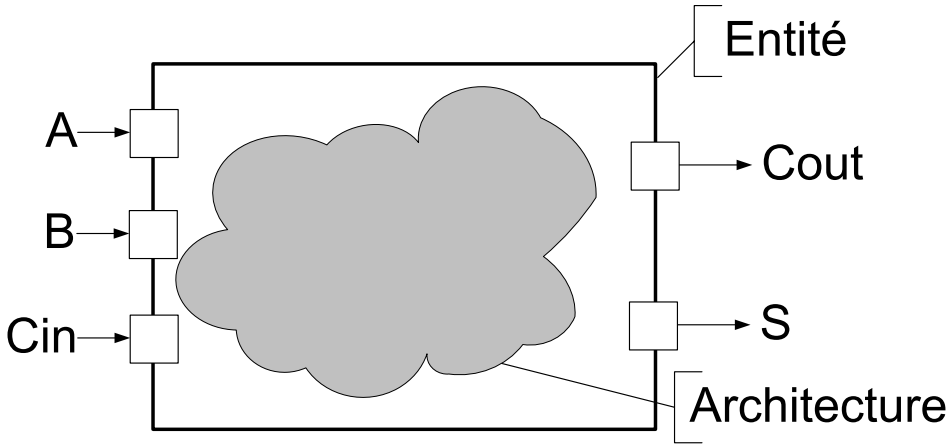
Les modes d'entrée/sortie: exemple corrigé

L'exemple précédent ne fonctionne pas en simulation, il manque un reset.

```
entity exemple is
port ( clk,rst: in std_logic; a_i,b_i: in std_logic; c_o: out std_logic);
end exemple;

architecture behave of exemple is
signal c_int: std_logic;
begin
    c_o <= c_int;
    process(clk,rst)
    begin
        if rst = '1' then
            c_int<='0';
        elsif rising_edge(clk) then
            if a_i = b_i then
                c_int <= not c_int;
            end if;
        end if;
    end process;
end behave;
```

Entités et architectures

Exemple: additionneur de 1 bit



Spécification d'entité

```
entity full_adder is
    port (
            a_i:     in  std_logic;
            b_i:     in  std_logic;
            c_in_i:  in  std_logic;
            s_o:     out std_logic;
            c_out_o: out std_logic);
end full_adder;
```

Architectures

- Flot de données: équations booléennes
- Comportementale: algorithme
- Structurelle: système composé de sous-blocs, analogue à l'écriture d'une netlist d'un design schématique

Architecture: flot de données

Additionneur: flot de données

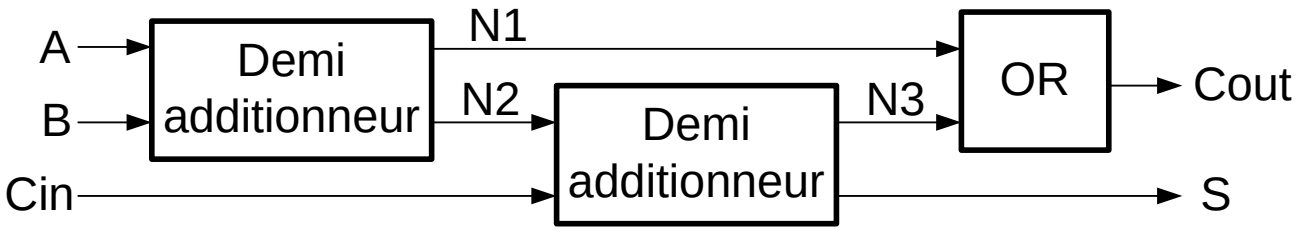
```
architecture dataflow of full_adder is

    signal inter: std_logic;

begin
    inter   <= a_i xor b_i;
    s_o     <= inter xor c_in_i;
    c_out_o <= (a_i and b_i) or (inter and c_in_i);
end dataflow;
```

Architecture: comportementale

a_i	b_i	c_in_i	s_o	c_out_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
architecture behave of full_adder is
begin
    process(a_i, b_i, c_in_i)
        variable i: integer;
        constant table_s: std_logic_vector(0 to 3):="0101";
        constant table_cout: std_logic_vector(0 to 3):="0011";
    begin
        i := 0;
        if a_i = '1'    then i := 1;     end if;
        if b_i = '1'    then i := i + 1; end if;
        if c_in_i = '1' then i := i + 1; end if;
        s_o     <= table_s(i);
        c_ou_ot <= table_c_out_o(i);
    end process;
end behave;
```

Architecture: description structurelle




```
architecture struct of full_adder is

    component half_adder is
        port(a_i,b_i; in std_logic; s_o, c_out_o: out std_logic);
    end component;

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

    signal n1, n2, n3: std_logic;

begin
    c1: half_adder port map(a_i, b_i, n2, n1);
    c2: half_adder port map(n2, c_in_i, s_o, n3);
    c3: or_gate port map(n1, n3, c_out_o);
end structure;
```

Architecture: description structurelle (2)

```
begin
    c1: half_adder
    port map(
        a_i      => a_i,
        b_i      => b_i,
        s_o      => n2,
        c_out_o => n1
    );

    c2: half_adder
    port map(
        a_i      => n2,
        b_i      => c_in_i,
        s_o      => s_o,
        c_out_o => n3
    );

    c3: or_gate
    port map(
        a_i => n1,
        b_i => n3,
        c_o => c_out_o
    );
```

Evite de se tromper

Signaux et ports

- Une entité est vue de l'extérieur comme une boite noire
- On ne voit que ses ports
  - Correspondent à des *fils*
- A l'intérieur d'une architecture les fils sont représentés par des signaux: `signal`
  - Chaque connexion que vous avez dans un schéma correspond à un signal dans une architecture VHDL

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 scalaires: entiers

Type prédéfini dans le paquetage standard

```
type INTEGER is range -2'147'483'648 to 2'147'483'647;
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
```

Exemple

```
type profondeur_de_pile is range 1 to 12;
subtype profondeur_de_pile is POSITIVE range 1 to 12;
```

Flottants

```
type nouveau_flottant is range 1.23 to 5.43;
```

Types scalaires: physique

Type physique: caractérisation d'un objet par son unité de base, l'intervalle de ses valeurs et ses éventuelles sous-unités.

Exemple: type time

```
type time is range -(2**63-1) to (2**63+1)
-- 64 bits nécessaires pour exprimer l'heure en femtosecondes

units
    fs;                 -- unité de base
    ps  = 1000 fs;
    ns  = 1000 ps;
    us  = 1000 ns;
    ms  = 1000 us;
    sec = 1000 ms;
    min = 60 sec;
    hr  = 60 min;
end units;
```

Types scalaires: énuméré

Type énuméré: caractérisation d'un objet par la liste complète de ses valeurs

Types énumérés prédéfinis

```
type bit            is ('0','1');          -- caractères
type boolean        is (false,true); -- identificateurs
type character      is 256 caractères du jeu ISO 8859-1;
type severity_level is (NOTE,WARNING,ERROR,FAILURE);
```

Exemples de types et sous-types non prédéfinis

```
type logic4  is ('0','1','X',Z'); -- surcharge avec le type bit
type state_t is (IDLE,INIT,CHECK,DONE);
type mixed_t is (FALSE,'1','2',IDLE); -- surcharge avec types boolean,
                                      --  bit et states
```

Attention

```
logic4'('1')  ≠ bit'('1')
state_t'(IDLE) ≠ mixed_t'(IDLE)
```

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

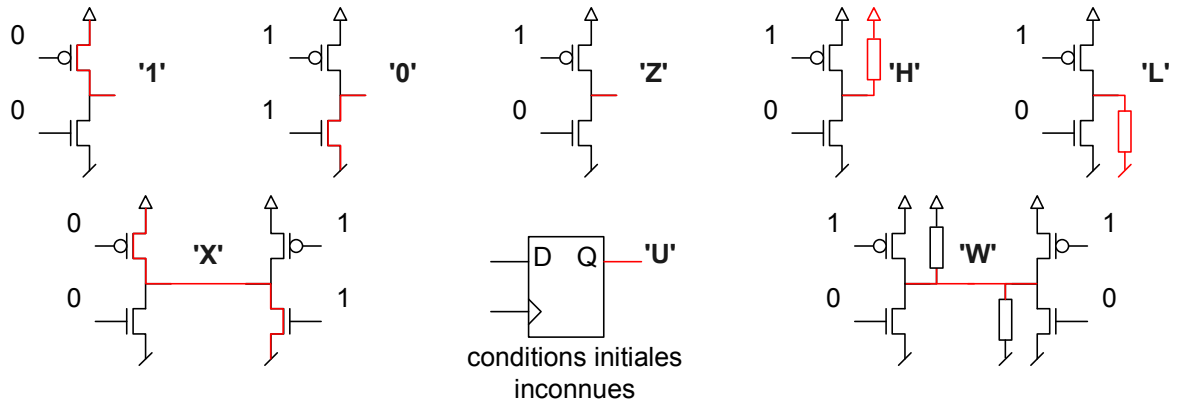
Exemple

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

signal state, next_state : state_t;

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

Types scalaires énuméré: std_logic

Le type `std_logic` est largement utilisé pour la simulation et la synthèse des systèmes matériels

```
type std_ulogic is ( 'U', -- état non initialisé
                     'X', -- état logique indéfini fort
                     '0', -- état logique 0 fort
                     '1', -- état logique 1 fort
                     'Z', -- état à haute impédance
                     'W', -- état logique indéfini faible
                     'L', -- état logique 0 faible
                     'H', -- état logique 1 faible
                     '-', -- état logique indifférent );
std_logic: subtype std_logic is resolved std_ulogic;
```



Exemple d'une 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 result: r_bit := 'Z';
begin
    for i in sources'range loop
        case sources(i) is
            when 'X' => return 'X';
            when '0' => if result = '1' then
                            return 'X'; else result := '0';
                        end if;
            when '1' => if result = '0' then
                            return 'X'; else result := '1';
                        end if;
            when 'Z' => null;
        end case;
    end loop;
    return result;
end r_resolution;
```

Types composites: tableaux

- Type tableau (array type): type composite consistant en un groupe d'objets dont le type est identique et la position indexée.

Exemple

```
type std_logic_vector is array (NATURAL range <>) of std_logic;
subtype word16        is std_logic_vector(15 downto 0);
type memory128        is array (127 downto 0) of word16;

signal memory   : memory128;
signal word     : mots16;
signal onebit   : std_logic;

word   <= memory(12);
onebit <= memory(12)(4);

type BIT_VECTOR is array(NATURAL range <>) of BIT;

signal onevector: BIT_VECTOR(5 downto 0);
```

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

Types composites: signed/unsigned

- Deux types de la librairie ieee.numeric_std permettent de manipuler des vecteurs comme des entiers signés ou non signés

Exemple

```
signal a : unsigned(7 downto 0);
signal b : unsigned(7 downto 0);
signal c : unsigned(7 downto 0);
signal d : signed(7 downto 0);
signal e : std_logic_vector(7 downto 0);
signal f : integer;

c <= a + b;
e <= std_logic_vector(a);
e <= std_logic_vector(d);
a <= unsigned(e);
d <= signed(e);
a <= to_unsigned(e,8);
f <= to_integer(a);
```

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
- Sera aussi utilisé avec les transactions

Exemple

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

Type accès/access

Le type `access` correspond à un pointeur en C.

Exemple

```
type     int_pointer_t is access integer;   -- pointeur sur un entier

variable a, b: int_pointer_t;

a      := new integer'(18);
b      := new integer;
b      := a;
b.all  := 25;
a.all   = ?
```

Attention, les pointeurs ne sont pas synthétisables.

Type accès/access

- Les pointeurs seront utilisés en simulation
- Typiquement pour gérer des FIFOs virtuelles

Exemple

```
type stimulus_t is record
    date   : time;
    valeur : bit;
end record;
type pointer_stimulus_t is access stimulus_t; -- pointeur sur un stimulus

variable point: pointer_stimulus_t;

point        := new stimulus_t(10 ns,'1');
point.valeur := '0';
```

Attention, les pointeurs ne sont pas synthétisables.

Type fichier/FILE

- Sera traité dans le cadre de la vérification

Instructions concurrentes

- Les instructions concurrentes d'affectation correspondent à des processus implicites.
- Utilisées en mode *flot de données*

Affectations

```
a <= b or c;
a <= b or c after 5 ns;
a <= transport b or c after 5 ns;
```

attention à l'espace avant ns

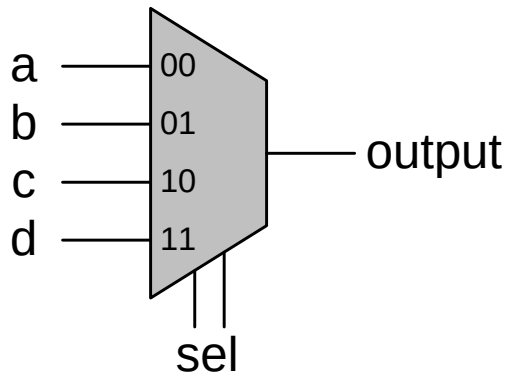
Instruction concurrente conditionnelle

Syntaxe

```
[label: ] nom_signal <= [mode_delai]
                        { forme_onde when expression_booléenne else}
                        forme_onde [when expression_booléenne];
```

Exemple: multiplexeur

```
output <= a when sel="00" else
          b when sel="01" else
          c when sel="10" else
          d;
```



Exemple: processus équivalent

```
process(sel,a,b,c,d)
begin
    if sel="00" then
        output <= a;
    elsif sel="01" then
        output <= b;
    elsif sel="10" then
        output <= c;
    else
        output <= d;
    end if;
end process;
```

Affectation sélective

Syntaxe

```
[label:] with expression select
    nom_signal <= [transport] forme_onde1 when choix1,
                              forme_onde2 when choix2,
                              ...
                              forme_ondeN when choixN;
```

Exemple: multiplexeur

```
with sel select
output <= a when "00",
          b when "01",
          c when "10",
          d when others;
```

Processus équivalent

```
process(sel,a,b,c,d)
begin
  case sel is
  when "00"   => output<=a;
  when "01"   => output<=b;
  when "10"   => output<=c;
  when others => output<=d;
  end case;
end process;
```

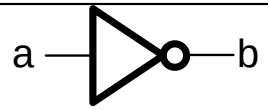
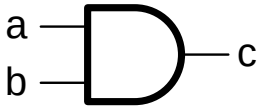
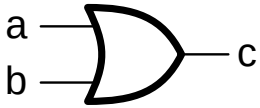
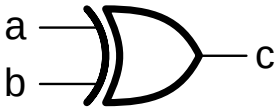
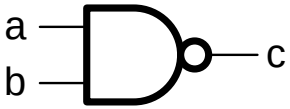
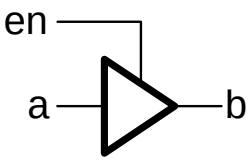
Du VHDL au schéma

- Lorsque l'on décrit un système en VHDL, il faut:
  - Etre conscient du schéma qui sera généré par le synthétiseur!
- Les composants de base sont relativement faciles à identifier
- Pour des descriptions structurelles, la décomposition est évidente
- Pour des descriptions comportementales, cela dépend de l'écriture...

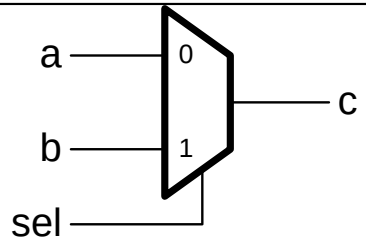
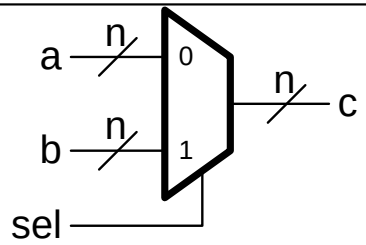
Composants de base

- Composants de base des designs à réaliser au laboratoire:
- Combinatoires
  - Portes logiques: and, or, xor, nand, ...
  - Porte tri-state
  - Multiplexeurs
  - fonctions de type additionneur, soustracteur, comparateur
- Séquentiels
  - Bascules D (avec ou sans load)
  - Registres

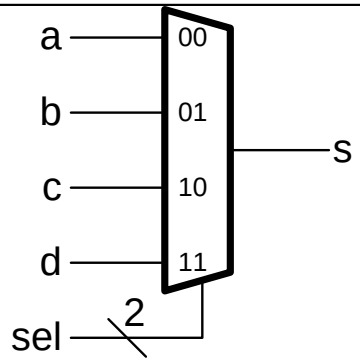
Composants combinatoires

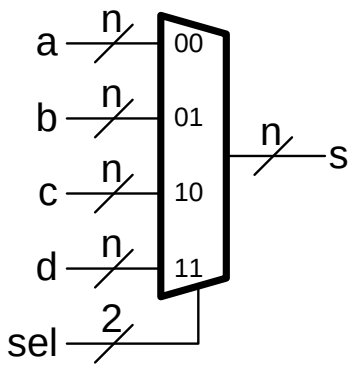
Not		`b <= not a`
And		`c <= a and b`
Or		`c <= a or b`
Xor		`c <= a xor b`
Nand		`c <= a nand b`
...	...	...
Tri-state		`c<= a when en='1' else 'Z'`

Composants combinatoires: multiplexeurs (1)

mux2		`c <= a when sel='0' else b`
mux2_n		`c <= a when sel='0' else b`

Composants combinatoires: multiplexeurs (2)

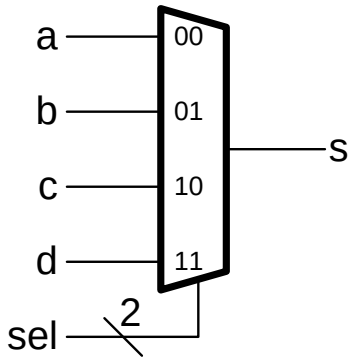
mux4		`s <= a when sel="00" else`
		`b when sel="01" else`
		`c when sel="10" else`
		`d;`

mux4_n		`s <= a when sel="00" else`
		`b when sel="01" else`
		`c when sel="10" else`
		`d;`

- ... mux8, mux16, ...

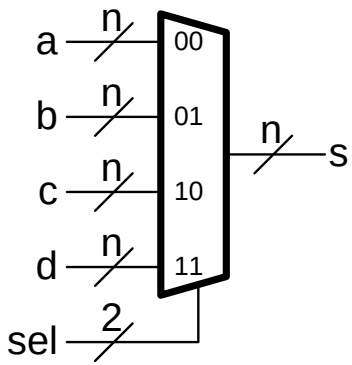
Composants combinatoires: multiplexeurs (3)

mux4



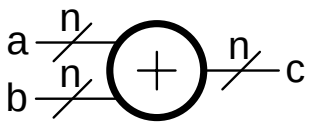
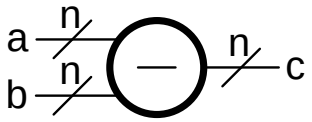
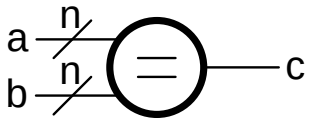
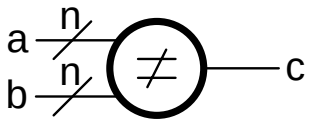

```
s <= with sel select
        a when "00",
        b when "01",
        c when "10",
        d when others;
```

mux4_n



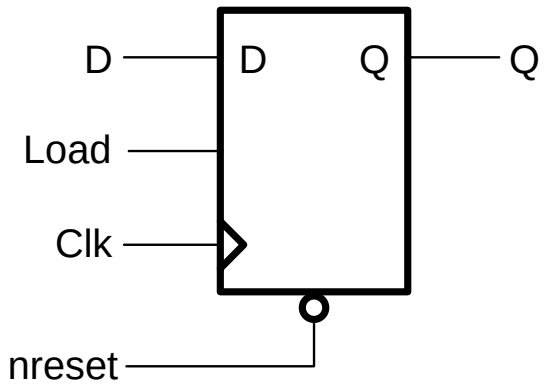

```
s <= with sel select
        a when "00",
        b when "01",
        c when "10",
        d when others;
```

Composants combinatoires: opérations

$+$		`c <= a + b` (librairie nécessaire)
$-$		`c <= a - b` (librairie nécessaire)
$=$		`c <= '1' when a = b else '0'`
$\neq$		`c <= '0' when a = b else '1'`
...	...	...

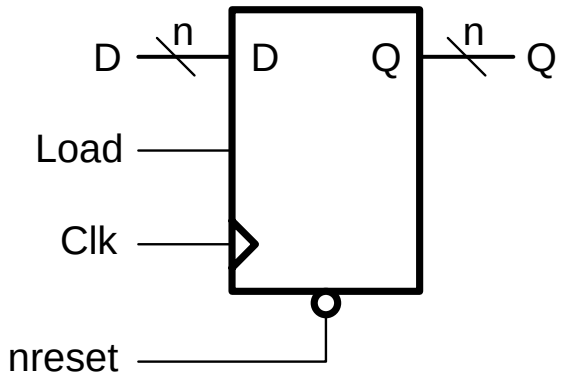
Composants séquentiels

Bascule D




```
process(clk,nReset) is
begin
    if (nReset='0') then
        Q <= '0';
    elsif (rising_edge(clk) then
        if (load='1') then
            Q <= D;
        end if;
    end if;
end process;
```

Registre



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

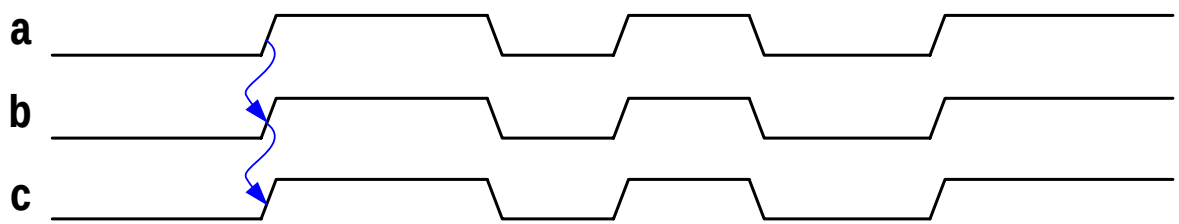
Comportement des signaux

- VHDL concurrent:

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

≅

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



- VHDL séquentiel équivalent:

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

Attention mauvaise pratique

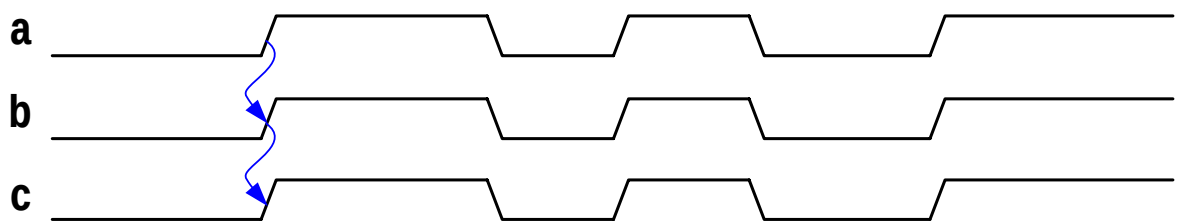
Comportement des signaux: mauvais exemple

- VHDL séquentiel:

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

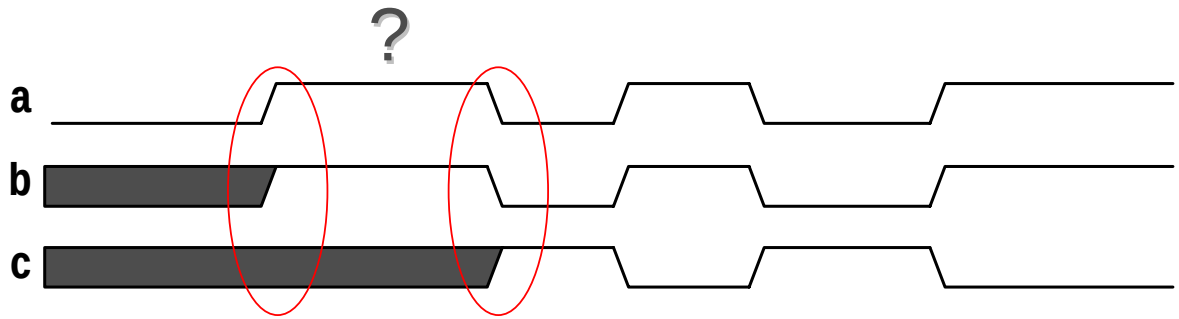
$\simeq$

```
c <= a;
```



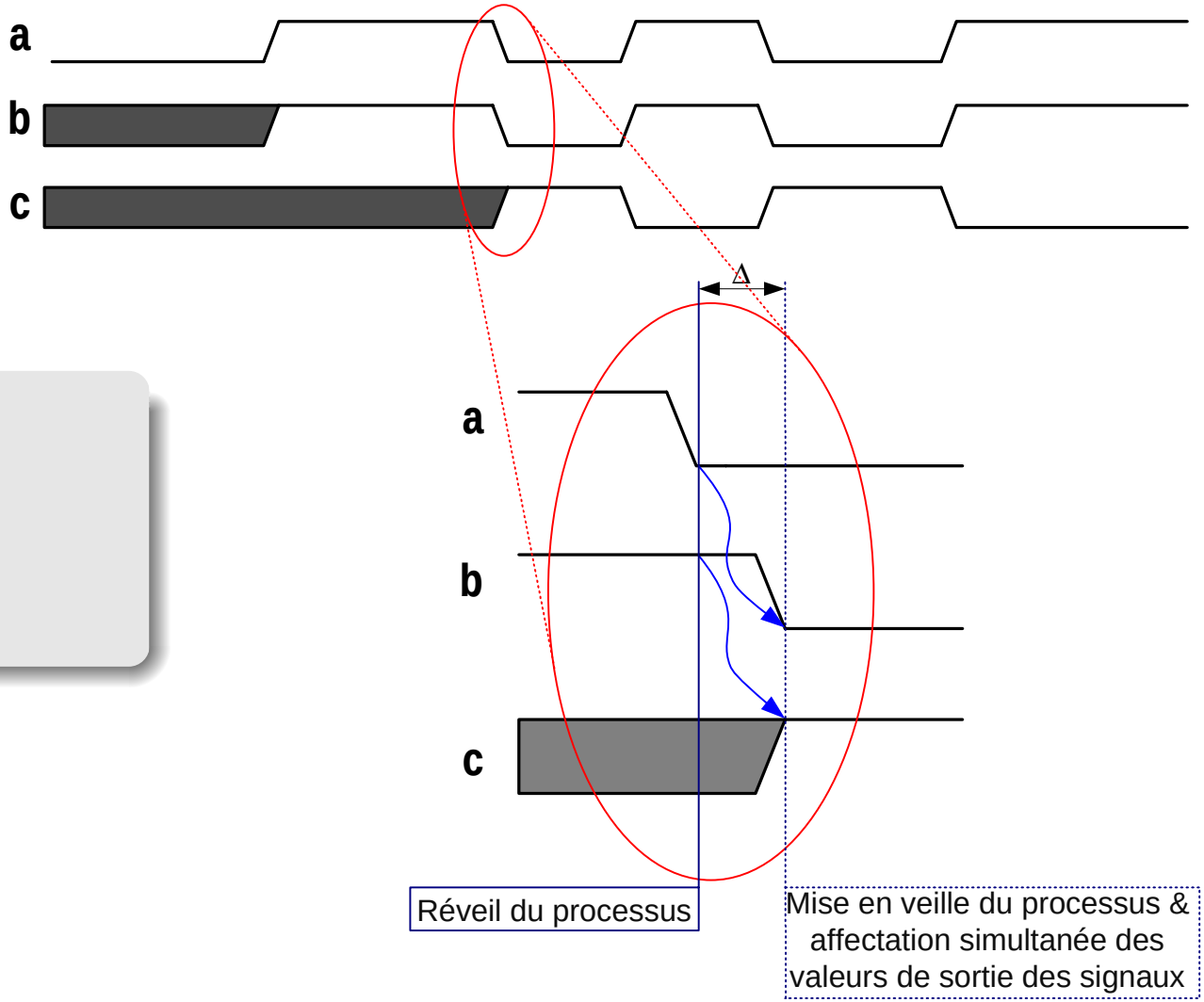
- VHDL séquentiel (mauvais exemple):

```
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 , b)
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 ...`

VHDL: processus combinatoires, règles à respecter

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

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(...) 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

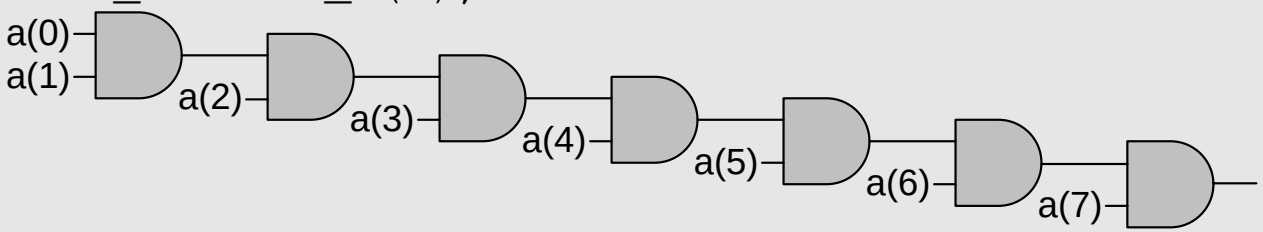
```
entity and_8 is
    port (a_i : in  std_logic_vector(7 downto 0);
          z_o : out std_logic);
end and_8;

architecture behave of and_8 is
begin
   process(a_i)
   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(a_i)
   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;
```

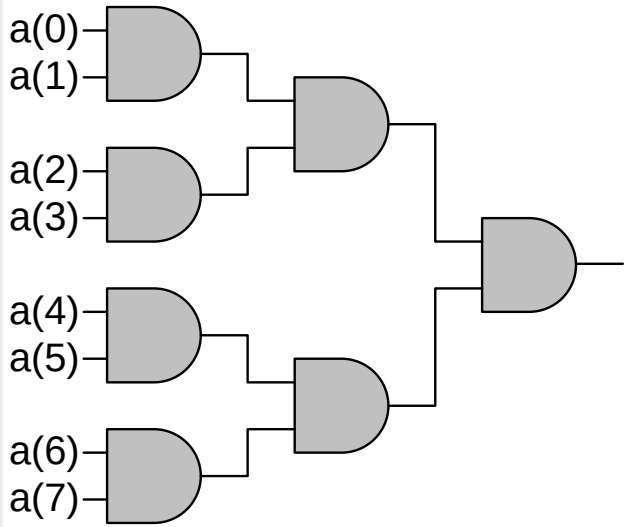


Utilisation des variables

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(a_i)
   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;
```

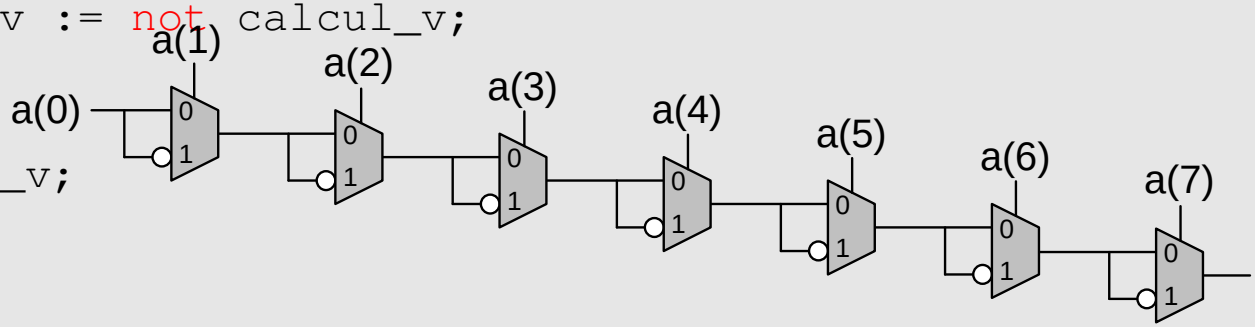
Solution optimisée:



Utilisation des variables

Exemple: détecteur de parité

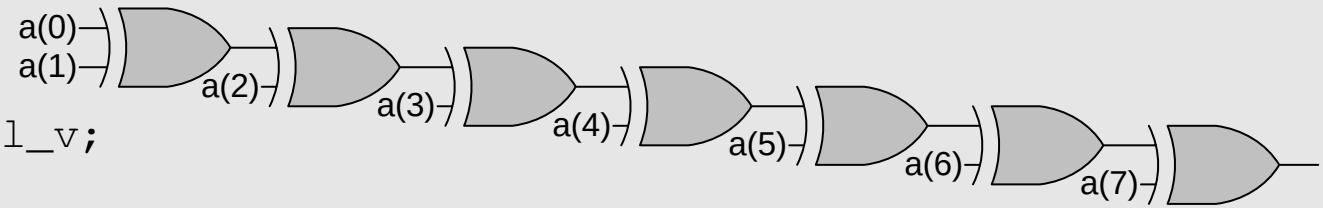
```
entity parity is
    port (a_i : in  std_logic_vector(7 downto 0);
          z_o : out std_logic);
end parite;

architecture behave of parity is
begin
   process(a_i)
   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(a_i)
   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;
```

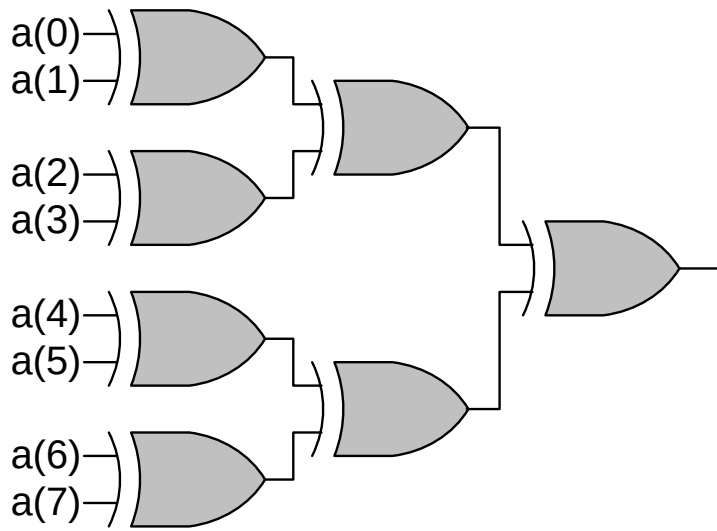


Utilisation des variables

Exemple: détecteur de parité

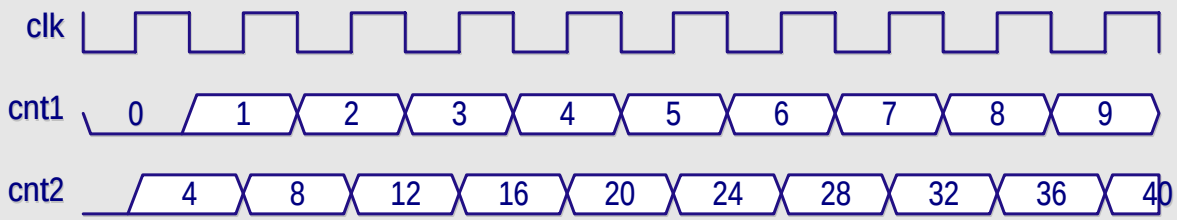
```
entity parity is
    port (a_i : in
std_logic_vector(7 downto 0);
          z_o : out std_logic);
end parity;

architecture behave of parity is
begin
   process(a_i)
   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:



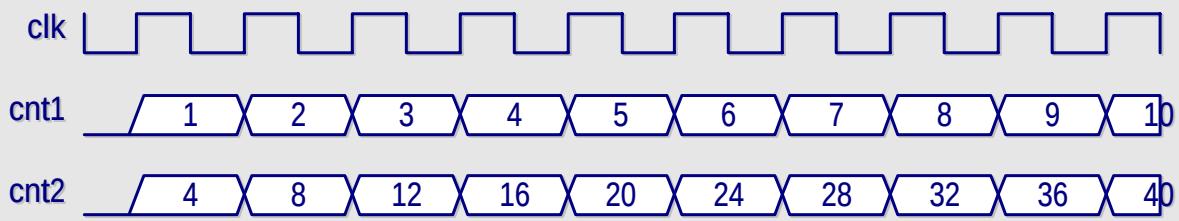
Comportement des signaux et variables (1)

```
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;
end behave1;
```



Comportement des signaux et variables (2)

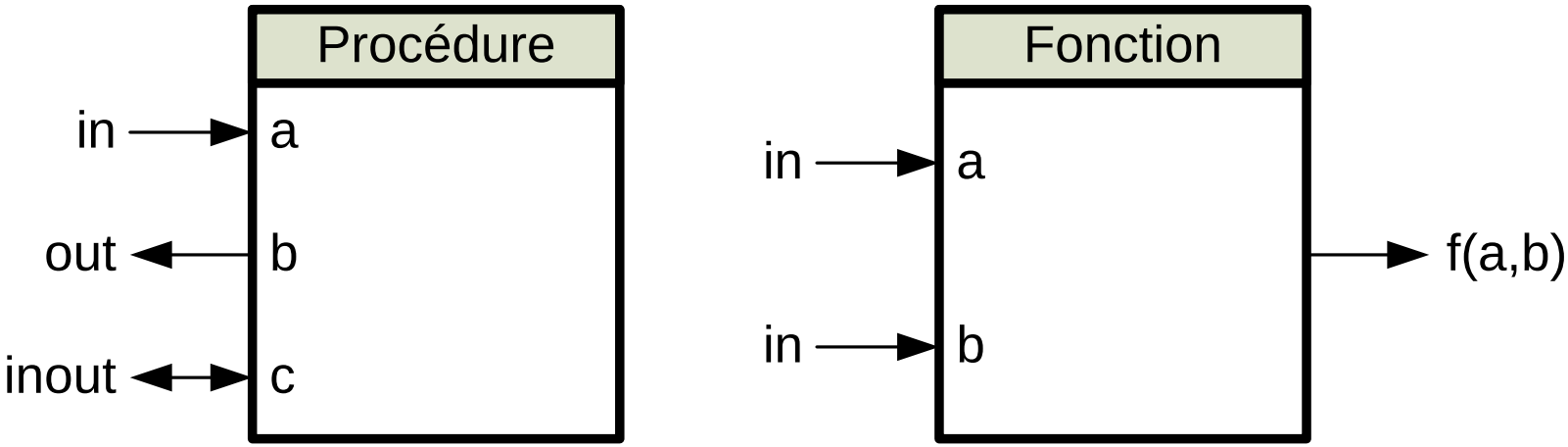
```
entity test is
port ( clk : in std_logic;
       cnt1 : out unsigned (7 downto 0);
       cnt2 : out unsigned (7 downto 0));
end test;

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;
end behave2;
```



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
```

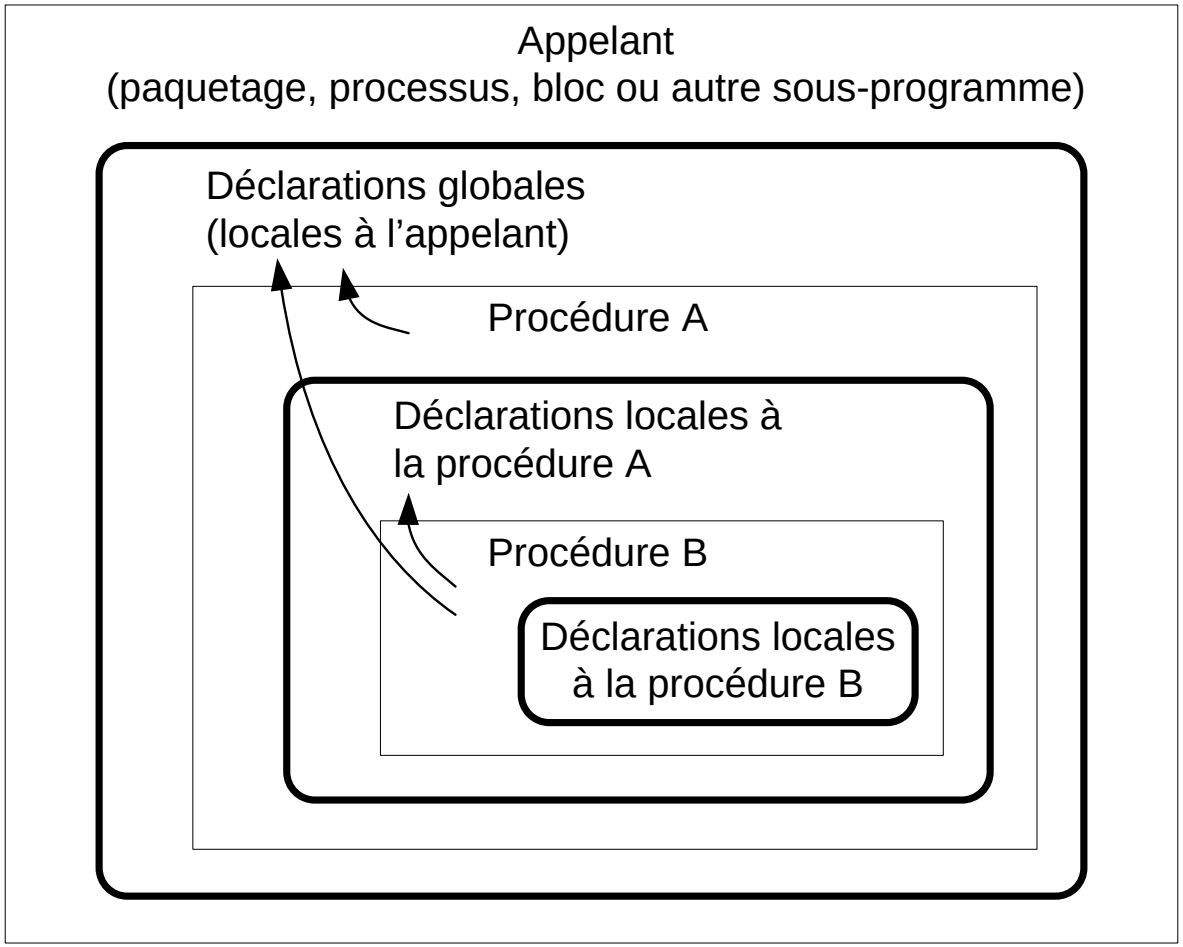
Exemple: Fonction de résolution

- 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



Décomposition

- Un système complexe doit être décomposé en sous-systèmes
- Les sous-systèmes décomposés en sous-systèmes plus petits
- ... Jusque à des blocs "simples"
- Pour ce faire: instanciation de composants
- Un composant est vu comme une boîte noire ayant des entrées et des sorties
- Mot réservé: `component`

Instanciation (1)

- Un composant qui doit être instancié doit être déclaré
  - Dans la partie déclarative de l'architecture de l'"instanciateur"
  - Dans un paquetage
- Les ports et génériques sont déclarés de la même manière que dans l'entité

Exemple de déclaration dans une architecture

```
architecture struct of my_system is

    component my_sub_system is
    port (
        clk_i  : in  std_logic;
        rst_i  : in  std_logic;
        val_i  : in  std_logic_vector(7 downto 0);
        pair_o : out std_logic
    );
    end component;

begin
    ...
end struct;
```

Instanciation (2)

- L'instanciation se fait en connectant les ports à des signaux/ports

Exemple de déclaration dans une architecture

```
architecture struct of my_system is

    signal one_value_s : std_logic_vector(7 downto 0);
    signal parity_s    : std_logic;

begin
    my_comp: my_sub_system
    port map (
        clk_i  => clk_i, -- clk_i est un port de my_system
        rst_i  => rst_i,
        val_i  => one_value_s, -- signal interne
        pair_o => parity_s
    );
end struct;
```

Instanciation (2)

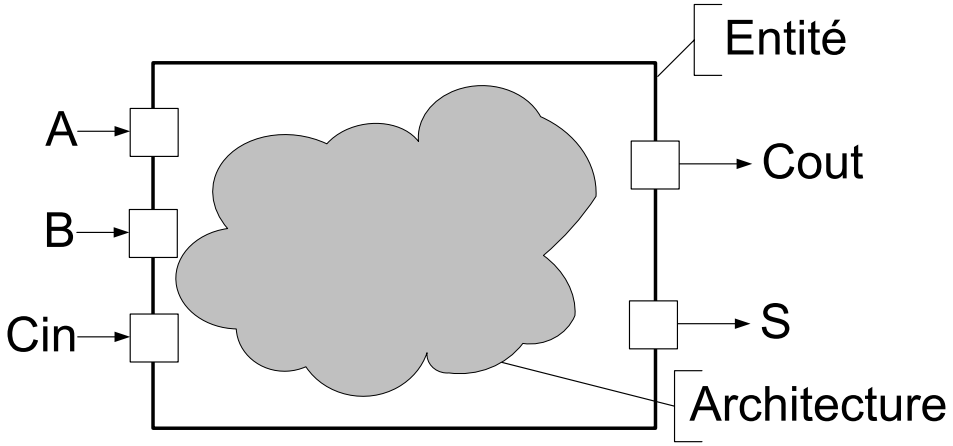
- Si paramètres génériques...

Exemple avec paramètre générique

```
architecture struct of my_system is

begin
    mon_comp: mon_sous_systeme
    generic map (TAILLE => 8) -- pas de ; à la fin...
    port map (
        clk_i  => clk_i, -- clk_i est un port de mon_systeme
        rst_i  => rst_i,
        val_i  => une_valeur_s, -- signal interne
        pair_o => parite_s
    );
end struct;
```

Instanciation: Exemple

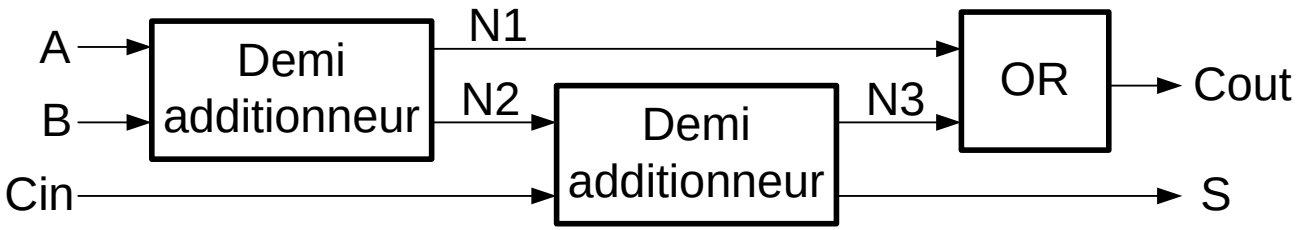
Exemple: additionneur de 1 bit



Spécification d'entité

```
entity full_adder is
    port (
        a_i:     in  std_logic;
        b_i:     in  std_logic;
        c_in_i:  in  std_logic;
        s_o:     out std_logic;
        c_out_o: out std_logic
    );
end full_adder;
```

Architecture: description structurelle




```
architecture struct of full_adder is

    component half_adder is
        port(a_i,b_i; in std_logic; s_o, c_out_o: out std_logic);
    end component;

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

    signal n1_s,n2_s,n3_s: std_logic;

begin
    c1: half_adder port map(a_i,b_i,n2_s,n1_s);
    c2: half_adder port map(n2_s,c_in_i,s_0,n3_s);
    c3: OR_gate port map(n1_s,n3_s,c_out_o);
end struct;
```

Architecture: description structurelle (bonne pratique)

- Il est mieux de spécifier le nom du port au lieu de compter sur l'ordre des signaux...

```
architecture struct of full_adder is
    ...
begin
    c1: half_adder
    port map(a_i => a_i,
             b_i => b_i,
             s_o => n1_s,
             c_o => n2_s);

    c2: half_adder
    port map(a_i => n2_s,
             b_i => c_in_i,
             s_o => n3_s,
             c_o => s_o);

    c3: OR_gate
    port map(a_i => n1_s,
             b_i => n3_s,
             c_o => c_out_o);
```

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(input_i)
        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(input0_i,input1_i,sel_i)
    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;
```

Generate

Syntaxe (forme conditionnelle)

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

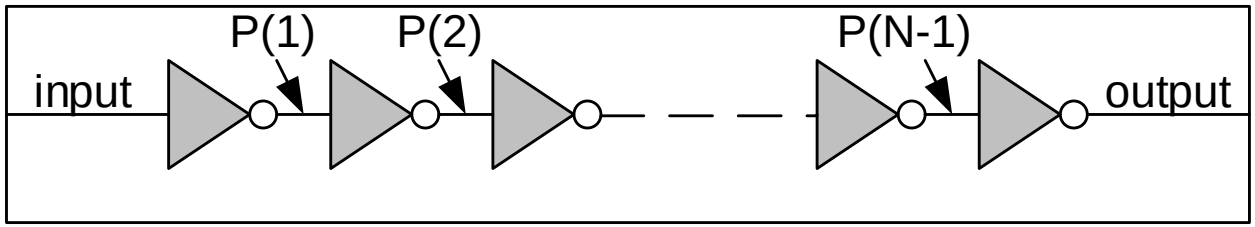
Syntaxe (forme itérative)

```
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: 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

Paquetage Numeric_std: Contexte

Exemple

```
signal a, b : std_logic_vector(7 downto 0);

...

a <= "11111111";
b <= "00000000";
if ( a < b ) then
    ...
else
    ...
end if;
```

- Que signifie cette comparaison?

- Fait partie de la bibliothèque IEEE
- Fait partie de la norme IEEE-1076.3, 1997
- But:
  - Offrir des types numériques et fonctions arithmétiques pour la synthèse
  - Permettre d'éviter des confusions arithmétiques

Déclaration

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

- Le paquetage définit 2 types numériques:
  - `unsigned`: vecteur représentant un nombre non signé
  - `signed`: vecteur représentant un nombre signé
- Ces deux types sont des nombres *entiers* binaires
- Ils sont basés sur le type `std_logic`
- Le bit le plus à gauche:
  - Bit le plus significatif, MSB (Most Significant Bit)
- Représentation des nombres signés:
  - Représentation en complément à 2
- Le paquetage contient:
  - Définition des 2 types
  - Surcharge des opérateurs arithmétiques pour ces 2 types
  - Fonctions de conversion et d'adaptation

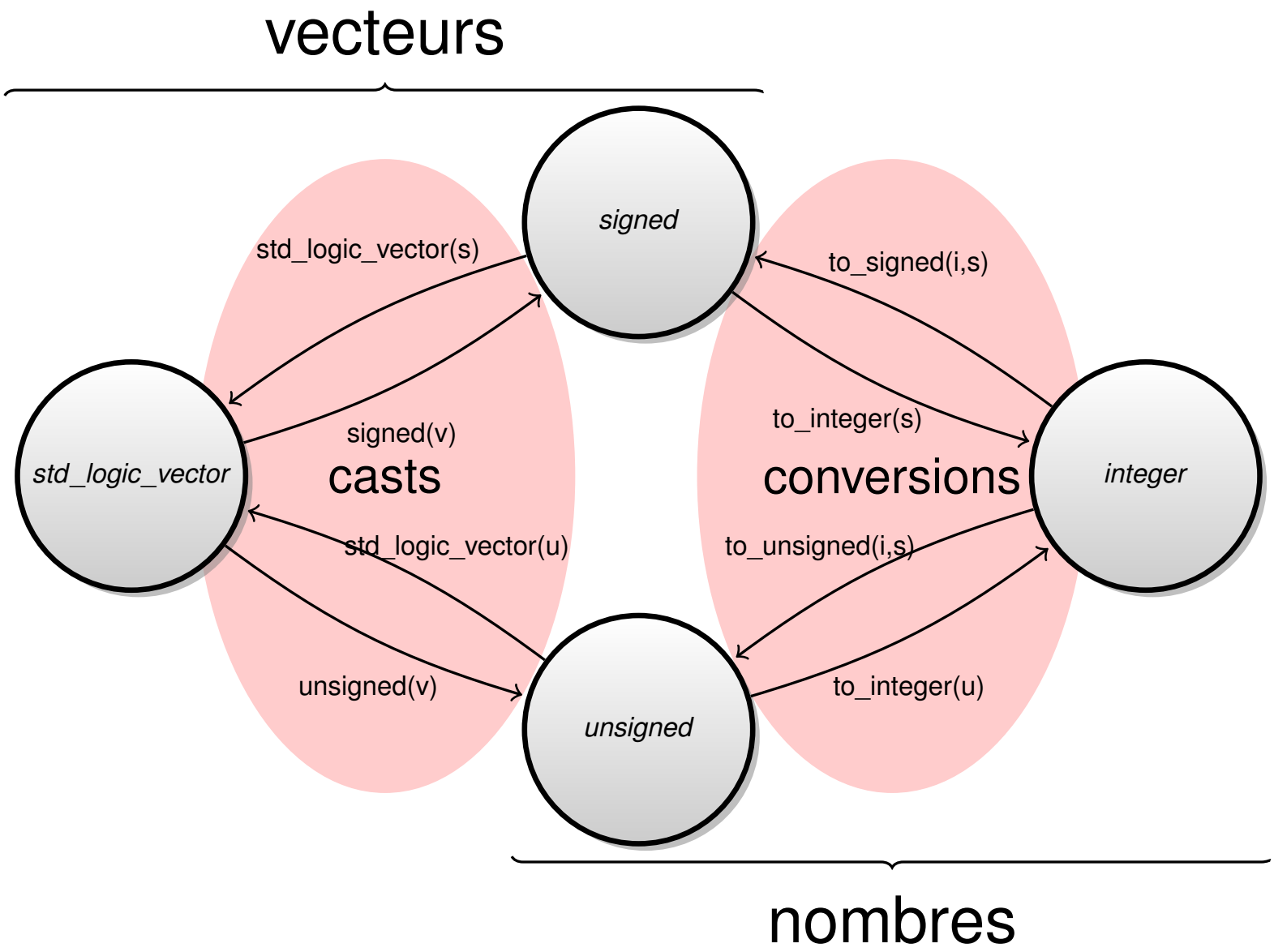
Types `Unsigned` et `Signed`

- Basés sur le type `std_logic`
- `std_logic_vector` versus `unsigned` ou `signed`:
  - Les types sont compatibles, mais
  - Spécifient une interprétation différente
    - `std_logic_vector`: vecteur binaire
    - `unsigned`: nombre entier positif en binaire
    - `signed`: nombre entier en binaire

Définitions

```
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
type SIGNED   is array (NATURAL range <>) of STD_LOGIC;
```

Conversions et casts



Adaptation de type (cast)

```
-- Déclaration des signaux :
signal vector : std_logic_vector(7 downto 0);
signal number : unsigned(7 downto 0);

-- Exemples d'adaptation de type (cast):
number <= unsigned(vector);
vector <= std_logic_vector(number);

-- Basé sur le std_logic, donc :
vector(i) <= number(i);   <-- correct
number(i) <= vector(i);   <-- correct
```

Fonctions de conversion (1)

Fonction `to_integer`

```
to_integer(arg: Unsigned) return natural;
to_integer(arg: Signed) return natural;
```

Fonction `to_unsigned`

```
to_unsigned(arg, size: natural) return unsigned;
```

Fonction `to_signed`

```
to_signed(arg, size: natural) return signed;
```

Fonctions de conversion (2)

Exemple

```
-- Déclaration des signaux :
signal nbr_entier : natural;
signal nbr_bin    : unsigned(7 downto 0);

--Exemples de conversion :
nbr_entier <= to_integer(nbr_bin);
nbr_bin    <= to_unsigned(nbr_entier,8);

-- en utilisant l'attribut : 'length
nbr_bin    <= to_unsigned(nbr_entier,nbr_bin'length);
```

Fonctions + et -

- Caractéristique intéressante:
  - 2 opérandes de tailles différentes

  ⇒ Résultat a la taille du plus grand

  ⇒ Result: `MAX(L'LENGTH,R'LENGTH)-1 downto 0`
- Combinaison des 2 opérandes (L et R):

```
(L, R: UNSIGNED)           return UNSIGNED
(L: UNSIGNED; R: NATURAL)  return UNSIGNED
(L: NATURAL; R: UNSIGNED)  return UNSIGNED
(L, R: SIGNED)             return SIGNED
(L: INTEGER; R: SIGNED)    return SIGNED
(L: SIGNED; R: INTEGER)    return SIGNED
```

Exemples

Exemples d'additions

```
-- Déclaration des signaux :
signal a, b : unsigned(7 downto 0);
signal sum  : unsigned(7 downto 0);

--Exemples d'addition :
sum <= a + b;
sum <= a + "0001";
sum <= b + 1;
sum <= a + "00110011";
sum <= b + '1';
```

Erreur!

Fonctions de comparaison

- Caractéristique intéressante:
  - Les opérandes peuvent être de tailles différentes
  - Résultat de type booléen
- Combinaison des 2 opérandes (L et R):

```
(L, R: UNSIGNED)           return BOOLEAN
(L: UNSIGNED; R: NATURAL)  return BOOLEAN
(L: NATURAL; R: UNSIGNED)  return BOOLEAN
(L, R: SIGNED)             return BOOLEAN
(L: INTEGER; R: SIGNED)    return BOOLEAN
(L: SIGNED; R: INTEGER)    return BOOLEAN
```

Exemples de comparaisons

Exemples de comparaisons

```
-- déclaration des signaux
signal a,b    : unsigned(3 downto 0);
signal result : boolean;

-- le résultat de la comparaison est un booléen
result <= (a = "1010");
result <= (a = 10);
result <= (a /= nSgn_B);
```

Exemples de comparaisons

```
-- déclaration des signaux
signal a, b   : signed(3 downto 0);
signal result : boolean;

-- le résultat de la comparaison est un booléen
result <= (a < "1100");
result <= (a < -4);
result <= (a > b);
```