Introduction au langage VHDL

VHSIC Hardware Description Language

Andres Upegui

Laboratoire de Systèmes Numériques
hepia

1. Introduction
2. Unités de conception
3. Instructions concurrentes
4. Les processus
5. Instructions séquentielles

Introduction

VHDL signifie VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. Ce langage de description matériel est:

- Un langage moderne, lisible, puissant et général mais complexe
- Une norme de l'IEEE (Institute of Electrical and Electronics Engineers)
- Un standard reconnu par tous les vendeurs d'outils CAO
- Un langage commercialement inévitable et technologiquement incontournable pour la description des systèmes matériels (en Europe)

Introduction

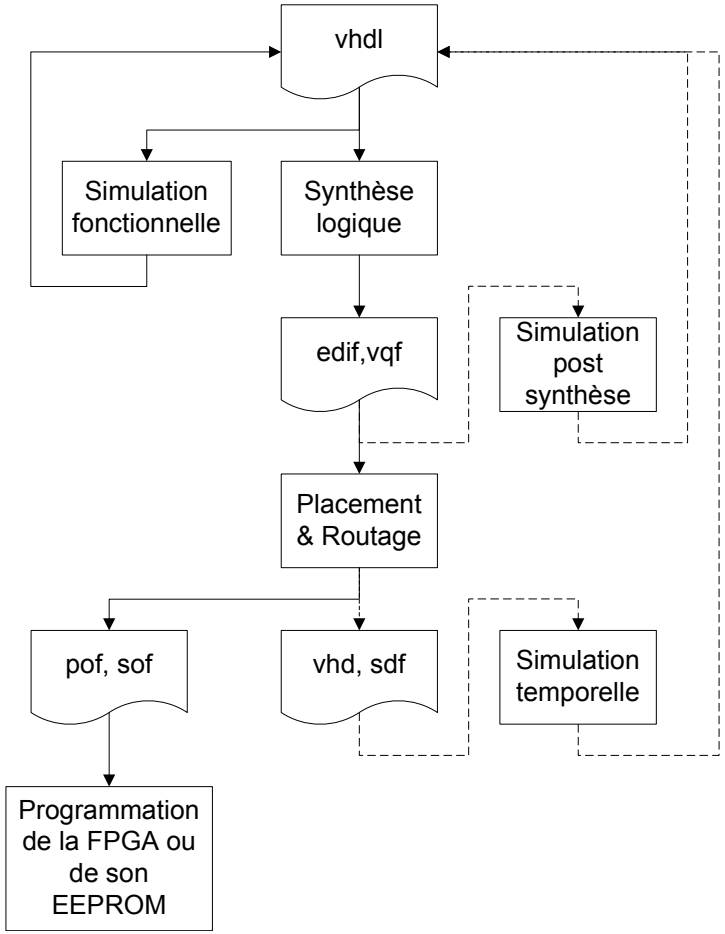
VHDL est exploité pour:

- Spécification de systèmes.
- Conception de systèmes (description de niveau transfert de registres).
- Simulation de systèmes.
- Cosimulation (interfaçage avec un autre langage, du type C/C++).
- Synthèse d'un système, en vue d'une implantation matérielle.

Histoire de VHDL

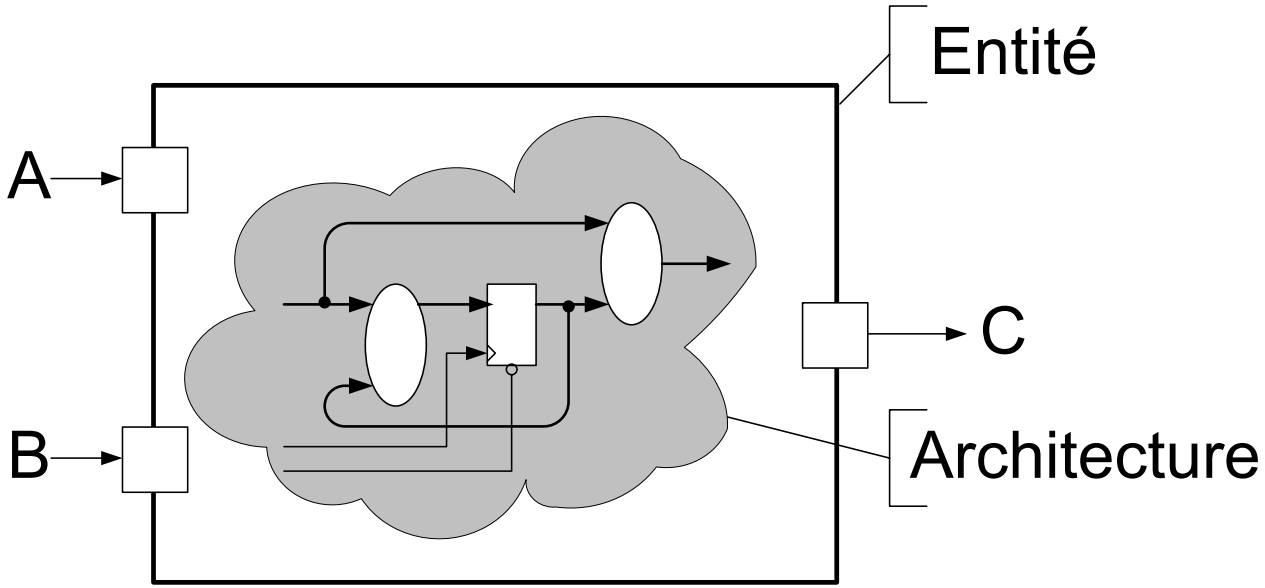
- 1980 Début du projet VHDL financé par le US DoD
- 1985 Première version 7.2 publique
- 1987 Première version du standard (IEEE Std 1076-1987)
- 1993 Mise à jour du standard (IEEE Std 1076-1993)
- 2002 Mise à jour du standard (IEEE Std 1076-2002)
- 2008 Mise à jour du standard (IEEE Std 1076-2008)

Flot de conception



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.



Entité de conception: exemple

Multiplexeur particulier

```
entity multiplexeur_part is
    port (
        a   : in    std_logic;
        b   : in    std_logic;
        sel : in    std_logic;
        c   : out   std_logic;
        cinv: out   std_logic
    );
end multiplexeur_part;


architecture flot_de_donnees of
multiplexeur_part is
signal sortie_mux: std_logic;
begin
    sortie_mux <= a when sel = '0'
                  else b;
    c          <= sortie_mux;
    cinv       <= not sortie_mux;
end flot_de_donnees;
```

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

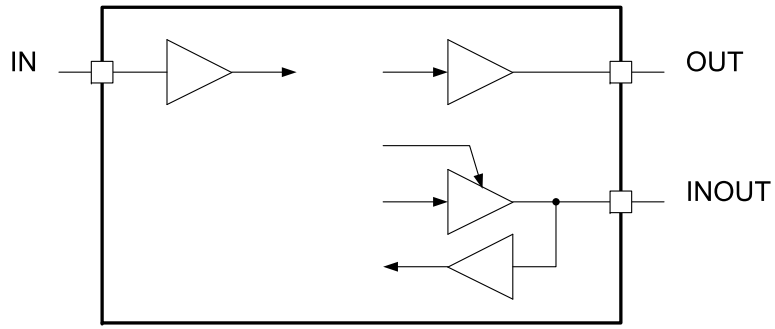
```
{clause_contexte} entity nom-entité is
    [generic (liste-paramètres);]
    [port (liste-ports);]
    [déclarations-locales]
[begin
    {instruction-concurrente-passive}]
end [entity] [nom-entité];
```

Corps d'architecture

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

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.



Les modes d'entrée/sortie: exemple

- Mais alors comment peut on lire une sortie?
- Utilisation de signaux internes comme suit:

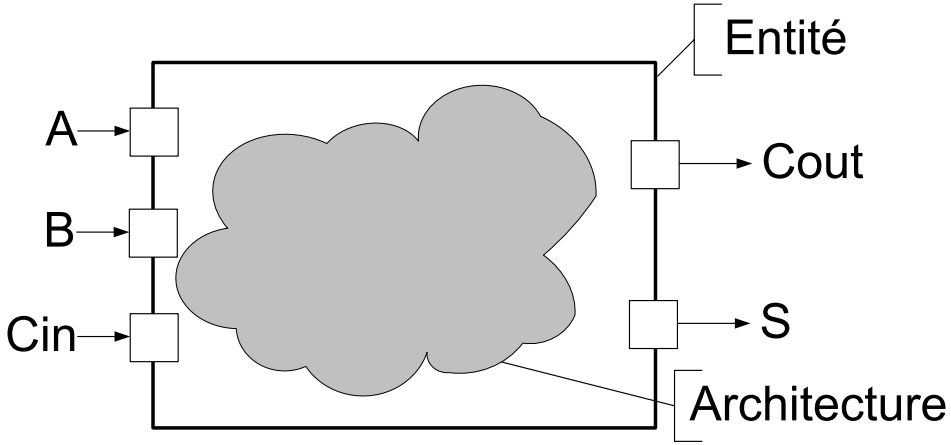
```
entity exemple is
port ( clk,rst: in std_logic; A,B: in std_logic; C: out std_logic);
end exemple;

architecture comp of exemple is
signal C_int: std_logic;
begin
    C<=C_int;
    process(clk,rst)
    begin
        if rst='1' then
            C_int<='0';
        elsif rising_edge(clk) then
            if A=B then
                C_int<=not C_int;
            end if;
        end if;
    end process;
end comp;
```

Entités et architectures

Exemple: additionneur de 1 bit

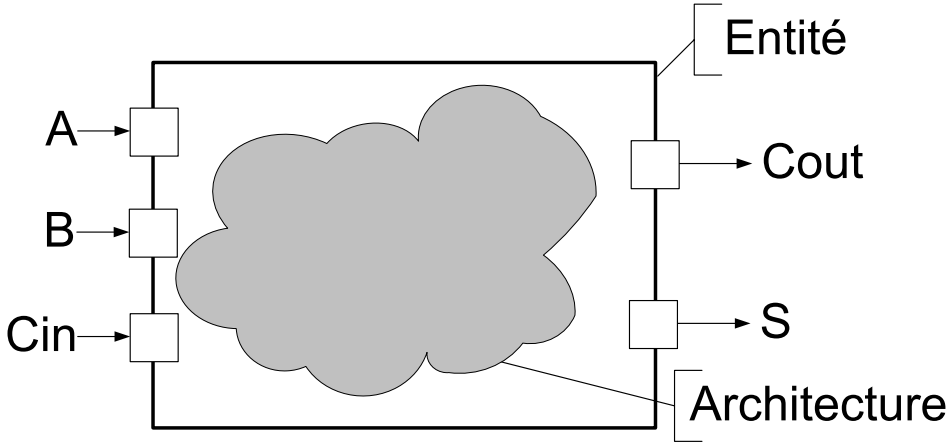
Entités et architectures

Exemple: additionneur de 1 bit



Entités et architectures

Exemple: additionneur de 1 bit



Spécification d'entité

```
entity additionneur is
    port (
            A:    in std_logic;
            B:    in std_logic;
            Cin:  in std_logic;
            S:    out std_logic;
            Cout: out std_logic);
end additionneur;
```

Architectures

Architectures

- Flot de données: équations booléennes

Architectures

- Flot de données: équations booléennes
- Comportementale: algorithme

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

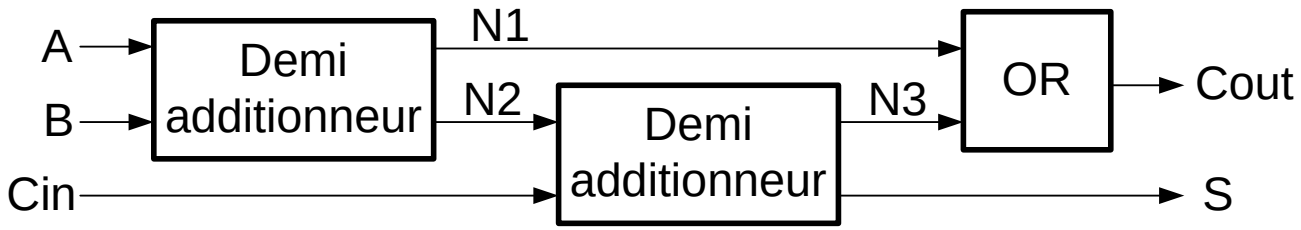
```
architecture flot_de_donnees of additionneur is

    signal inter: std_logic;

begin
    inter <= A xor B;
    s     <= inter xor Cin;
    Cout  <= (A and B) or (inter and Cin);
end flot_de_donnees;
```

Architecture: comportementale

A	B	Cin	S	Cout
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 comportement of additionneur is
    signal somme : std_logic_vector(1 downto 0);
begin

    process (A,B.Cin)
    begin
       if A='1' and B='1' and Cin='1' then
          somme <= "11";
       elsif (A='1' and B='1')  or
             (A='1' and Cin='1') or
             (B='1' and Cin='1') then
          somme <= "10";
       elsif A='1' or B='1' or Cin='1' then
          somme <= "01";
       else somme <= "00";
       end if;
    end process;

    S <= somme(0);
    Cout <= somme(1);

end comportement;
```

Architecture: description structurelle




```
architecture structure of additionneur is

    component demi_additionneur is
        port(A,B; in std_logic; S, Cout: out std_logic);
    end component;

    component porte_OU is
        port(A,B: in std_logic; C: out std_logic);
    end component;

    signal N1,N2,N3: std_logic;

begin
    C1: demi_additionneur port map(A,B,N1,N2);
    C2: demi_additionneur port map(N2,Cin,N3,S);
    C3: porte_OU port map(N1,N3,Cout);
end structure;
```

Instructions concurrentes

- Les instructions concurrentes d'affectation correspondent à des processus implicites.
- Trois types d'affectations concurrentes:
  - Affectation inconditionnelle
  - Affectation conditionnelle
  - Affectation sélective

Affectation inconditionnelle

```
a <= b or c;
a <= b and c and d;
a <= (b and c) or (b and d);
```

Instruction concurrente conditionnelle

Syntaxe

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

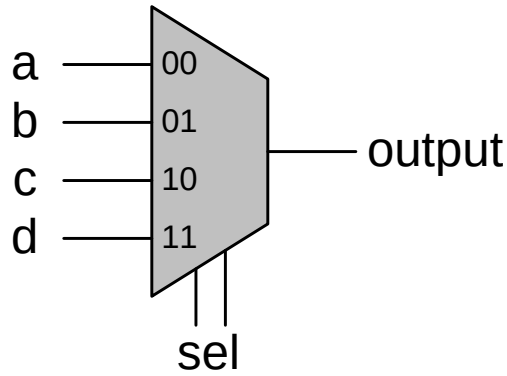
Instruction concurrente conditionnelle

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;
```



Affectation sélective

Syntaxe

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

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;
```

Exercices: instructions concurrentes

Décrire à l'aide du langage VHDL les composants suivants:

- Un décodeur à 2 entrées et 4 sorties
- Le même décodeur avec un signal de enable (si enable = '0' toutes les sorties restent à '0').
- Un encodeur à 4 entrées et 2 sorties.
- Un additionneur 2-bits avec retenue en entrée et en sortie.
  - entrées: A1, A0, B1, B0, Cin
  - sorties: S1, S0, Cout
  - pensez à utiliser des signaux internes.

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;
```

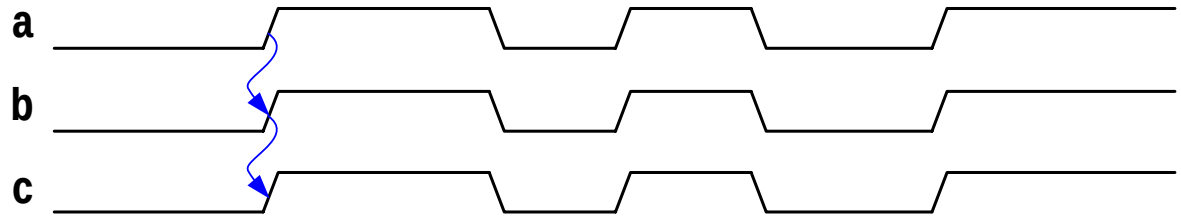
Comportement des signaux

- VHDL concurrent:

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

$\cong$

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



- VHDL séquentiel équivalent:

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

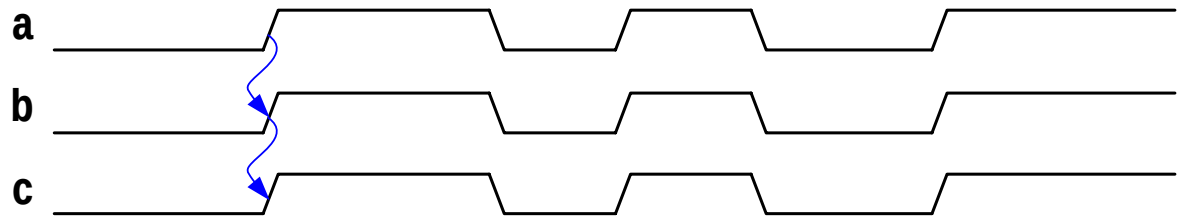
Comportement des signaux: mauvais exemple

- VHDL séquentiel:

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

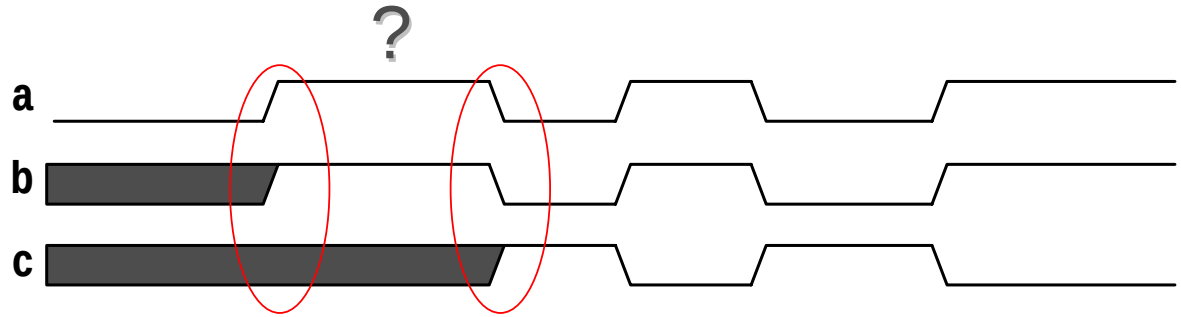
$\simeq$

```
c<=a;
```

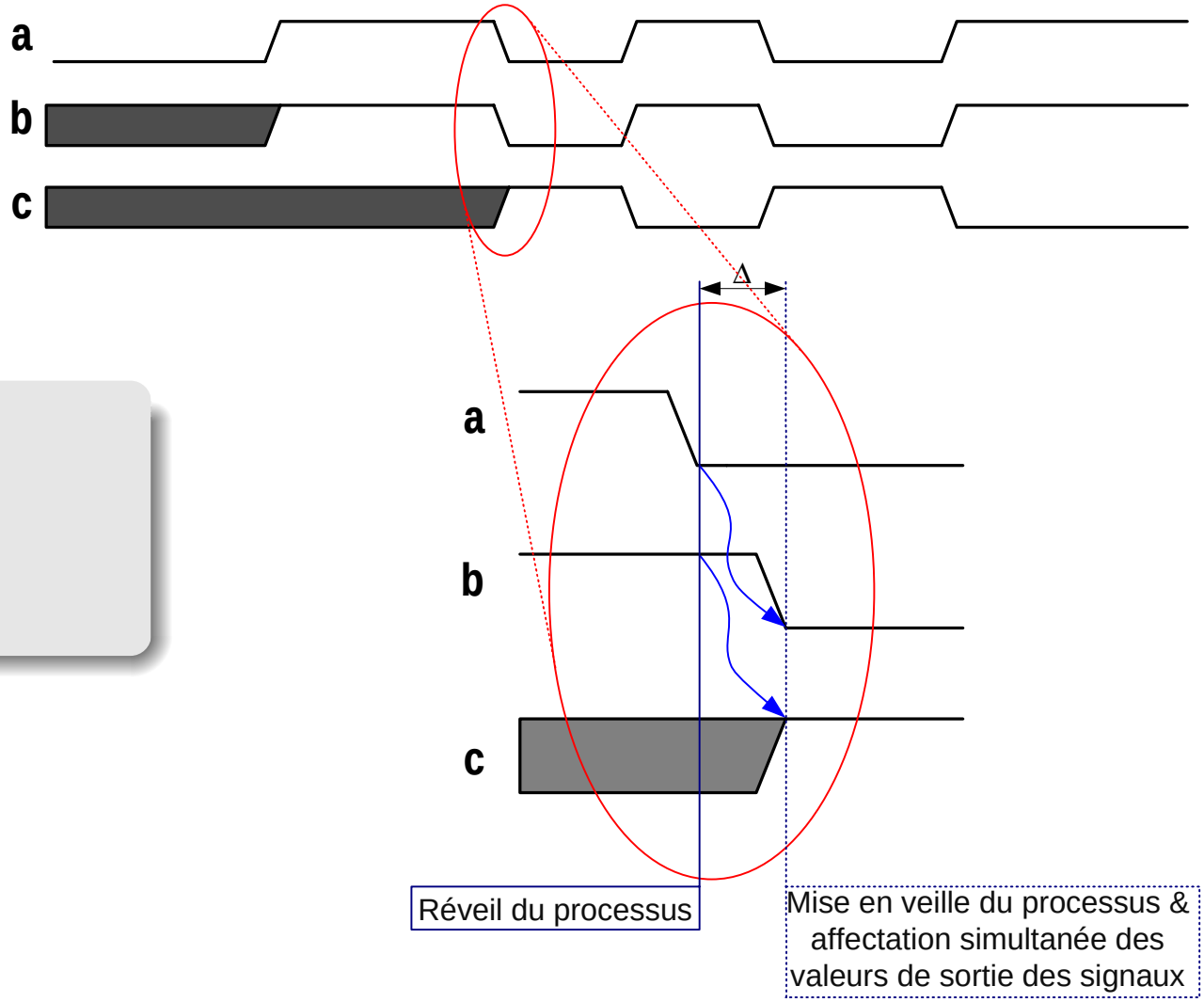


- VHDL séquentiel (mauvais exemple):

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



Comportement des signaux: mauvais exemple détaillé




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

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 etat is ...

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 is (Init,Fetch,Writeback,Done);
signal s: state;
...

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

Exemple 2

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

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

Exercices: instructions séquentiels

Décrire à l'aide d'un process les composants suivants:

- Un décodeur à 2 entrées et 4 sorties avec un signal de enable.
- Un multiplexeur à 4 entrées et 1 sortie.
- Un détecteur de palindromes avec 7 bits d'entrée.
- Un comparateur sur 3 bits:
  - entrées: A2, A1, A0, B2, B1, B0
  - sorties: lt (lower than), gt (greater than), eq (equal)
- Fonction majorité sur un vecteur de taille 7.
- Calculateur de parité sur un vecteur de taille 8.