



TP 02 : Initiation aux instructions concurrentielles (structure de contrôle)

Thème : Multiplexeur, Démultiplexeur, ALU

Objectifs pédagogiques

Ce TP vise à :

- Comprendre les trois styles d'affectation en VHDL :
L'affectation sélective (with...select)
L'affectation conditionnelle structurée (case...when)
L'affectation conditionnelle compacte (when....else)
- Appliquer ces styles à deux circuits combinatoires :
Le multiplexeur (MUX)
Le démultiplexeur (DEMUX)
Unité arithmétique et logique (ALU)
- Développer une méthode rigoureuse pour écrire, tester et simuler un programme VHDL synthétisable.

Partie théorique : Les styles d'affectation en VHDL

♦ 1. Affectation sélective (with...select)

Ce style permet d'associer une valeur à **une seule sortie** selon un signal de sélection (**obligatoire**). Il est utilisé **hors processus**, directement dans l'architecture.

Avantages :

- Syntaxe concise et lisible
- Idéal pour les multiplexeurs simples
- Permet de couvrir tous les cas du sélecteur avec

Structure générale :

```
with <sélecteur> select  
  <signal> <= <valeur1> when <cas1>,  
              <valeur2> when <cas2>,  
              ...  
              <valeurN> when others;
```

♦ 2. Affectation conditionnelle structurée (case...when)

Ce style permet de tester les différentes valeurs d'un sélecteur et d'affecter des signaux en fonction de chaque cas (**peut avoir plusieurs sortie**). Il est utilisé **dans un processus**, il est plus flexible et recommandé pour les circuits évolutifs.

Avantages :

- Permet d'initialiser les sorties
- Lisible et extensible
- Adapté aux démultiplexeurs et aux décodages

Structure générale :

```
case <sélecteur> is
  when <cas1> => <signal> <= <valeur1>;
  when <cas2> => <signal> <= <valeur2>;
  ...
  when others => <signal> <= <valeurN>;
end case;
```

♦ 3. Affectation conditionnelle compacte (when...else)

Ce style est utilisé pour des conditions simples à deux cas (**signal de sélection pas obligatoire**). Il permet une écriture rapide et directe. Si tu veux piloter **plusieurs sorties en même temps** pour une même sélection, tu dois écrire **plusieurs instructions when ... else**, une par sortie. Il est utilisé **hors processus**

Avantages :

- Très utile pour les multiplexeurs 2 vers 1
- Syntaxe minimale
- Facile à lire pour des sélections binaires

Structure générale :

```
<signal> <= <valeur1> when <condition> else
    <valeur2> when <condition2> else
    <valeur3> when <condition2> else
    <valeur4>;
```

Étapes pour écrire un programme VHDL

- ♦ Étape 1 : Définir l'entité
 - Identifier les entrées : signaux de données, signal de sélection
 - Définir les sorties : signal unique (MUX) ou plusieurs sorties (DEMUX)
 - Choisir les types standards : std_logic, std_logic_vector
- ♦ Étape 2 : Choisir le style d'affectation
 - Utiliser (with....select) pour des sélections claires et directes
 - Utiliser (case...when) ou (if...then) pour des logiques structurées avec initialisation
 - Utiliser (when...else) pour des conditions binaires simples
- ♦ Étape 3 : Écrire les instructions d'affectation
 - Placer les instructions dans l'architecture
 - Couvrir tous les cas du sélecteur
 - Vérifier la cohérence des types et des valeurs
 - Écrire le code VHDL du multiplexeur 2→1 de 2 bits. En utilisant l'affectation sélective (with...select).
 - Écrire le code VHDL du démultiplexeur 1→2 de 2 bits. En utilisant l'affectation conditionnelle structurelle (when...else).
- ♦ Étape 4 : Compiler et corriger
 - Compiler le code dans Quartus II
 - Corriger les erreurs de syntaxe ou de typage

- Vérifier que le code est synthétisable.
- ◆ Étape 5 : Étapes pour simuler le programme
 - Créer un fichier de test .vwf dans Quartus II
 - Ajouter les signaux d'entrée (sel, a, b, din, etc.)
 - Définir des séquences de test pour chaque valeur de sélection
 - Observer les sorties et vérifier la logique.
- ◆ Étape 6 : PIN Planner

Associer les signaux logiques du projet VHDL (ex d0 ,d1 ,sel ,y) aux broches physiques du FPGA/CPLD afin de relier correctement le design au matériel (boutons, LEDs, switch...).

 - Pour le **MUX** : Utiliser les switch (1..4) Pour introduire les données, le bouton 1 pour sélection Et les LED (1..2) pour la sortie.
 - Pour le **DEMUX** : Utiliser les switch (1..2) Pour introduire les données, le bouton 1 pour sélection Et les LED (1..4) pour la sortie.
- ◆ Étape 7 : Programmer
 - Dans la fenêtre Programmer, cocher **Program/Configure**.
 - Cliquer sur **Start**.
 - Attendre le message **Success**.

Conception d'une Unité Arithmétique et Logique (UAL) en VHDL

Objectifs pédagogiques

- Comprendre le rôle d'une **UAL (ALU)** dans un système numérique.
- Apprendre à décrire un même comportement en VHDL avec **plusieurs styles d'affectation** :
 - Affectation conditionnelle (when ... else)
 - Affectation sélective (with ... select)
 - Instruction case dans un process
- Développer la capacité à comparer les styles de description et à choisir le plus adapté selon le contexte.
- Simuler et vérifier le fonctionnement de l'UAL.

Partie théorique

- ◆ Qu'est-ce qu'une ALU ?
 - Une **ALU** est un bloc combinatoire qui réalise des opérations arithmétiques (addition, soustraction) et logiques (ET, OU, XOR, NON).
 - Elle reçoit en entrée :
 - **Deux opérandes** (A et B).
 - **Un code d'opération** (OP) qui sélectionne l'opération à effectuer.
 - Elle fournit en sortie : **Le résultat** de l'opération.

◆ Codage des opérations

Exemple de codage possible pour OP (2 bits) :

OP : std_logic_vector (2 downto 0);

OP (0) = 0 (A + B), 1 (A – B). OP(1) = 0 (A AND B), 1 (A OR B)

OP(2) = 0 (résultat de l'unité logique), 1 (résultat de l'unité arithmétique)

 Étapes pour écrire un programme VHDL

◆ Étape 1 : Création du projet

- Créer un nouveau projet Quartus II.
- Nommer l'entité : ALU_VHDL.
- Définir les ports :
A : opérande 1 (vecteur binaire 2 bits). (**Switch 1..2**)
B : opérande 1 (vecteur binaire 2 bits). (**Switch 3..4**)
OP : code de commande (3 bits selon le nombre d'opérations). (**Bouton 1..3**)
Result : résultat de l'opération sélectionnée (**LED 1..2**)

◆ Étape 2 : Écriture du code

1. Pour l'unité arithmétique écrire le code avec **affectation case**.
2. Pour l'unité logique écrire le code Avec **affectation sélective**.
3. Pour la sortie Avec **affectation conditionnelle**.

◆ Étape 3 : Simulation

- Simuler le code avec différents couples (A, B) et valeurs de OP.
- Vérifier que le résultat correspond à la table des opérations.

◆ Étape 4 : PIN Planner

Associer les signaux logiques du projet VHDL (ex a, b, op, result) aux broches physiques du FPGA/CPLD afin de relier correctement le design au matériel (boutons, LEDs, switch...).

- Pour l'**ALU** : Utiliser les switch (1..4) Pour introduire les données A et B, le bouton 1..3 pour sélection l'opération Et les LED (1..2) pour la sortie.

◆ Étape 5 : Programmer

- Dans la fenêtre Programmer, cocher **Program/Configure**.
- Cliquer sur **Start**.
- Attendre le message **Success**.

ANNEX

#-----LED-----#

set_location_assignment	PIN_67	-to	led[1]
set_location_assignment	PIN_66	-to	led[2]
set_location_assignment	PIN_61	-to	led[3]
set_location_assignment	PIN_58	-to	led[4]
set_location_assignment	PIN_57	-to	led[5]
set_location_assignment	PIN_56	-to	led[6]
set_location_assignment	PIN_55	-to	led[7]
set_location_assignment	PIN_54	-to	led[8]

#-----Bouton-----#

set_location_assignment	PIN_49	-to	key[1]
set_location_assignment	PIN_48	-to	key[2]
set_location_assignment	PIN_47	-to	key[3]
set_location_assignment	PIN_44	-to	key[4]
set_location_assignment	PIN_43	-to	key[5]
set_location_assignment	PIN_42	-to	key[6]

#-----Switch-----#

set_location_assignment	PIN_53	-to	ckey[1]
set_location_assignment	PIN_52	-to	ckey[2]
set_location_assignment	PIN_51	-to	ckey[3]
set_location_assignment	PIN_50	-to	ckey[4]



