



## TP 01 : Initiation à la Programmation avec VHDL sous Quartus II

Thème : Entité, architecture et simulation avec équations logiques

### Objectifs du TP

- Se familiariser avec la structure **entité/architecture** en VHDL.
- Décrire des circuits logiques simples (MUX, additionneur complet) **uniquement**
- Utiliser Quartus II pour la synthèse, la simulation, et la visualisation du comportement du circuit.
- **Sans testbench** (via l'éditeur de stimuli intégré).

### Partie théorique

#### ◆ Définition

Un multiplexeur est un circuit combinatoire qui permet de sélectionner une entrée parmi plusieurs et de la transmettre vers une sortie unique, selon la valeur d'un signal de sélection.

#### ◆ Multiplexeur 2→1 de 1 bits

- Entrées :
  - d0 : première entrée (**switch 1**)
  - d1 : deuxième entrée (**switch 2**)
- Sélecteur :
  - sel : bit de sélection (**bouton 1**)
- Sortie :
- y : sortie sélectionnée (**LED 1**)

#### ◆ Fonction logique

- $y = (d0 \cdot \overline{sel}) \text{ or } (d1 \cdot sel)$

#### ◆ Additionneur complet 1 bits

- Entrées :
  - a : première entrée (**switch 1**)
  - b : deuxième entrée (**switch 2**)
- Retenu d'entrée :
  - Cin : bit de retenue (**bouton 1**)
- Sortie :
  - sum : résultat de l'addition (**LED 1**)
  - cout : retenue (**LED 2**)

#### ◆ Fonction logique

- $sum = A \oplus B \oplus Cin$
- $cout = (A.B) + (A.Cin) + (B.Cin).$

## Préparation

- **Logiciel** : Quartus II (Project Navigator).
- **Bibliothèques** :  

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;
```
- **Organisation** :
  - Un fichier .vhd par composant.
  - Un fichier .vhd séparé pour le testbench.

## Étapes pratiques avec Quartus II

### ◆ Étape 1 : Création du projet

1. Ouvrir Quartus II.
2. Menu File → New Project Wizard.
3. Nommer le projet (ex. **mux2**).
4. Choisir le répertoire de travail.
5. Sélectionner la famille de FPGA utilisée (**MAX II** → **EPM570T100I5**).
6. Cliquez sur **device and pin option** → unused pins puis choisir **As input tri-stated**.
7. Valider.

### ◆ Étape 2 : Création du fichier VHDL

1. Menu File → New → VHDL File.
2. Écrire l'entité et l'architecture du multiplexeur.
3. Sauvegarder sous le nom **mux\_2to1.vhd**, même nom de l'entité.
4. Ajouter le fichier au projet via Project → Add/Remove Files in Project.

#### **Exemple 1 : Multiplexeur 2:1**

- Entity

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
entity mux_2to1 is  
port (  
  a : in std_logic;  
  b : in std_logic;  
  sel : in std_logic;  
  y : out std_logic );  
end mux_2to1;
```
  - Architecture (équation logique)

```
architecture logique of mux_2to1 is  
begin  
  y <= (a and (not sel)) or (b and sel);  
end logique;
```
- #### **Exemple 2 : Additionneur complet (Full Adder)**
- Entité

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_adder is
port (
a, b, cin : in std_logic;
sum, cout : out std_logic );
end full_adder ;

```

- Architecture (équations logiques)

```

architecture logique of full_adder is
begin
sum <= a xor b xor cin;
cout <= (a and b) or (a and cin) or (b and cin);
end logique;

```

#### ♦ Étape 3 : **Compilation**

1. Menu Processing → Start Compilation.
2. Vérifier qu'il n'y a pas d'erreurs de syntaxe.
3. Corriger si nécessaire.

#### ♦ Étape 4 : **Simulation (Waveform Editor)**

Sans testbench

1. Menu File → New → University program VWF File.
  2. Ajouter les signaux d0, d1, sel, y.
  3. Définir les valeurs de test pour d0, d1, sel.
  4. Menu Simulation → Run Functional Simulation.
  5. Observer la sortie y et vérifier qu'elle correspond à l'entrée sélectionnée.
- 👉 **Avantage** : rapide pour tester un petit composant.
  - 👉 **Limite** : non réutilisable, pas automatisé.

#### ♦ Étape 5 : **Analyse des résultats**

- Vérifier que les chronogrammes correspondent aux **tables de vérité** :  
**MUX2** :  $y = a$  si  $sel=0$ ,  $y = b$  si  $sel=1$ .  
**Full Adder** : sum et cout conformes aux équations logiques.

#### ♦ 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...).

##### ♦ Ouvrir le Pin Planner

1. Compiler une première fois le projet (même avec erreurs mineures).
2. Menu : Assignments → Pin Planner
3. Une fenêtre s'ouvre avec :
  - La liste des signaux du projet (colonne *Node Name*).
  - Les colonnes pour définir les broches (*Location*).
  - Un schéma du boîtier du FPGA.

◆ Identifier les signaux à assigner

- Les signaux déclarés en port dans ton entité VHDL apparaissent automatiquement.

Exemple :

```
entity mux_2to1 is
port (
d0 : in std_logic;
d1 : in std_logic;
sel : in std_logic;
y : out std_logic );
end mux_2to1;
```

- → Dans le Pin Planner, tu verras d0, d1, sel, y.

◆ Associer chaque signal à une broche physique

1. Dans la colonne **Location**, clique sur la cellule vide en face du signal.
2. Tape le nom de la broche (ex. PIN\_12) ou choisis-la dans le schéma du boîtier.
3. Répète pour chaque signal.

⚠ Les correspondances exactes (ex. d0 → PIN\_1) se trouvent dans le **schéma de ta carte** ou la **datasheet du FPGA**. **Voire Annex**

◆ Définir le standard électrique (I/O Standard)

- Dans la colonne **I/O Standard**, choisis par exemple :
  - 3.3-V LVTTTL (souvent utilisé pour CPLD/FPGA d'enseignement).
- Cela garantit la compatibilité avec les périphériques externes (LEDs, afficheurs, boutons).

◆ Sauvegarder et recompiler

1. Ferme le Pin Planner → Quartus sauvegarde les affectations dans le fichier .qsf.
2. Recompile le projet.
3. Vérifie dans le rapport de compilation que **toutes les broches sont bien assignées**.

🧠 Erreurs fréquentes des étudiants

- Oublier d'assigner une broche → Quartus met le signal en *floating*.
- Choisir une broche réservée (ex. alimentation, JTAG) → erreur de compilation.
- Ne pas respecter le bon standard I/O → risque de mauvais fonctionnement.

◆ Étape 7 : **Programmation du FPGA**

◆ Ouvrir Quartus Programmer

1. Menu : Tools → Programmer.
2. Vérifier que le câble **USB-Blaster** est détecté :
  - Hardware Setup → USB-Blaster [OK].

◆ Charger le fichier de programmation

- **Pour une programmation temporaire (volatile) :**

Ajouter le fichier .sof → le design reste actif tant que la carte est alimentée.

- **Pour une programmation permanente (non-volatile) :**  
ajouter le .pof → Le dessin Restera actif Même après Rallumage De la carte

◆ Programmer

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

 **Exercice : Conception d'un multiplexeur 2×1 de 2 bits en VHDL**

 **Objectif pédagogique**

Concevoir un multiplexeur à 2 entrées de 2 bits en VHDL, en respectant la séparation entité/architecture, et en utilisant une architecture logique simple. L'objectif est de comprendre le principe de sélection vectorielle et de simuler le comportement du circuit dans Quartus II.

 **Partie théorique**

◆ Définition

Un **multiplexeur** est un circuit combinatoire qui permet de **sélectionner une entrée parmi plusieurs** et de la transmettre vers une sortie unique, en fonction de la valeur d'un **sélecteur**.

◆ Multiplexeur 2×1 de 2 bits

- **Entrées :**
  - Deux vecteurs de 2 bits : d0 (**switch 1..2**), d1 (**switch 3..4**)
  - Un sélecteur sel (1 bits) (**bouton 1**)
- **Sortie :**
  - Un vecteur de 2 bits y (**LED 1..2**)

◆ Fonction logique

- Si sel = '0' → y = d0
- Si sel = '1' → y = d1

Chaque bit de la sortie est donc une combinaison logique des bits des entrées et des signaux du sélecteur.

◆ Applications

- Transmission de données sur un bus partagé
- Sélection d'une source parmi plusieurs périphériques
- Circuits arithmétiques et processeurs (choix d'opérandes)

 **Contexte technique**

Un **multiplexeur 2×1 de 2 bits** permet de sélectionner l'un des 2 vecteurs d'entrée (d0, d1) selon la valeur du sélecteur sel, et de le transmettre en sortie y(1 downto 0).

 **Étapes de conception en VHDL**

- ◆ Étape 1 : Analyse fonctionnelle
  - Identifier les **ports d'entrée et de sortie**.
  - Définir le rôle du sélecteur.
  - Établir la table de vérité reliant sel aux entrées sélectionnées.
- ◆ Étape 2 : Définition de l'entité
  - Déclarer les 2 entrées de 2 bits, le sélecteur de 1 bits, et la sortie de 2 bits.
  - Choisir le type `std_logic_vector`.
  - Écrire l'entité du multiplexeur 2×1 de 2 bits.
- ◆ Étape 3 : Architecture logique
  - Écrire les équations logiques de la sortie.
  - Écrire l'architecture logique avec sélection vectorielle.
- ◆ Étape 4 : Création du projet dans Quartus II
  1. Créer un nouveau projet.
  2. Ajouter un fichier VHDL pour le multiplexeur.
  3. Écrire l'entité et l'architecture.
  4. Compiler et corriger les erreurs éventuelles.
- ◆ Étape 5 : Simulation
  - Simuler university programme file.
  - Appliquer manuellement des valeurs binaires sur d0 à d1 et sel.
  - Observer la sortie y.
- ◆ 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...).
- ◆ Étape 7 : Programmer
  1. Dans la fenêtre Programmer, cocher **Program/Configure**.
  2. Cliquer sur **Start**.
  3. Attendre le message **Success**.

## **Exercice : Conception d'un démultiplexeur 1→2 de 2 bits en VHDL avec équations logiques**

### **Objectif pédagogique**

Concevoir un démultiplexeur à 2 sorties de 2 bits en VHDL, en utilisant uniquement des équations logiques pour distribuer un vecteur d'entrée vers l'une des 2 sorties selon un sélecteur. L'objectif est de comprendre le principe de distribution conditionnelle et de simuler le comportement du circuit dans Quartus II.

### **Partie théorique**

- ◆ Qu'est-ce qu'un démultiplexeur ?

Un **démultiplexeur** est un circuit combinatoire qui **redirige une seule entrée vers l'une des plusieurs sorties**, selon la valeur d'un **sélecteur**.

- Il est l'inverse d'un **multiplexeur**.
- Il permet de **distribuer** une donnée vers une destination choisie.
- ♦ Démultiplexeur 1→2 de 2 bits
  - **Entrée** : un vecteur D(1 downto 0) à transmettre. (**switch 1..2**)
  - **Sélecteur** : sel (**bouton 1**) détermine vers quelle sortie envoyer D.
  - **Sorties** : Y0 (**LED1..2**) , Y1 (**LED 3..4**) (chacune 2 bits).
- ♦ Fonction logique

Une seule des sorties reçoit la valeur de D, les autres sont forcées à "0000".

- Si sel = '0' → Y0 = D
- Si sel = '1' → Y1 = D
- ♦ Applications
  - Routage de données dans les bus.
  - Sélection de registres ou de périphériques.
  - Distribution conditionnelle dans les architectures séquentielles.

### Contexte technique

Un démultiplexeur 1→2 de 2 bits prend :

- Un vecteur d'entrée
- Un sélecteur
- Et transmet vers l'une des 2 sorties (Y0, Y1), toutes de 2 bits

Les autres sorties reçoivent «00 ».

### Étapes de conception en VHDL

- ♦ Étape 1 : Analyse fonctionnelle
  - **Entrées** : D(1 downto 0), sel.
  - **Sorties** : Y0, Y (chacune 2 bits)
  - **Comportement attendu** :
  - Si sel = '0' → Y0 = D, autres = "00"
  - Si sel = '1' → Y1 = D, etc.
- ♦ Étape 2 : Définition de l'entité
  - Déclarer les ports avec leurs directions et types.
  - Utiliser std\_logic\_vector.
  - Écrire l'entité du démultiplexeur 1→2 de 2 bits.
- ♦ Étape 3 : Architecture avec équations logiques
  - Écrire les équations pour chaque bit de chaque sortie.
  - Ne pas utiliser de process ni de with select.
- ♦ Étape 4 : Simulation dans Quartus II
  - Simuler university program file.
  - Appliquer des valeurs binaires sur D et sel.

- Observer quelle sortie reçoit D, les autres doivent être "0000".
- ◆ Étape 5 : 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...).
- ◆ Étape 6 : Programmer
  1. Dans la fenêtre Programmer, cocher **Program/Configure**.
  2. Cliquer sur **Start**.
  3. Attendre le message **Success**.



## Exercice : Conception d'un additionneur complet de deux bus de données de 2 bits en VHDL



### Objectif pédagogique

Concevoir un additionneur complet 2 bits en VHDL, en respectant la séparation entité/architecture, et en utilisant uniquement des équations logiques. L'objectif est de comprendre le fonctionnement d'un additionneur en cascade et de simuler son comportement dans Quartus II.



### Rappel théorique

- ◆ Un additionneur complet 1 bit prend en entrée :
  - Deux bits : A, B
  - Une retenue d'entrée : Cin
- ◆ Et produit :
  - Une somme : S
  - Une retenue de sortie : Cout
- ◆ Équations logiques :
  - Somme:  $S = A \oplus B \oplus Cin$ .
  - Retenue :  $Cout = (A.B) + (A.Cin) + (B.Cin)$ .

Pour un additionneur 4 bits, on cascade 4 additionneurs 1 bit avec propagation des retenues.



### Contexte technique

Un additionneur complet 2 bits permet d'additionner deux nombres binaires de 2bits, avec une retenue d'entrée (Cin) et une retenue de sortie (Cout). Il est constitué de **deux additionneurs 1 bits** connectés en cascade.



### Étapes de conception en VHDL

- ◆ Étape 1 : Analyse fonctionnelle
  - **Entrées :**
    - A(1 downto 0) : premier opérande (**switch 1..2**)

- B(1 downto 0) : second opérande (**switch 3..4**)
  - Cin : retenue d'entrée (**bouton 1**)
- **Sorties :**
- S(1 downto 0) : somme (**LED 1..2**)
- Cout : retenue finale (**LED3**)
- ◆ Étape 2 : Définition de l'entité
  - Déclarer les ports avec leurs directions et types.
  - Utiliser std\_logic\_vector pour les bus et std\_logic pour les bits.
- ◆ Étape 3 : Architecture logique
  - Utiliser les **équations logiques** :
  - Créer un signal interne C pour stocker les retenues intermédiaires.
  - C(0) représente la retenue **de la sortie** du premier additionneur de 1 bits.
- ◆ Étape 4 : Création du projet dans Quartus II
  1. Créer un nouveau projet VHDL.
  2. Ajouter un fichier adder4.vhd.
  3. Écrire l'entité pour deux bus de 2 bits.
  4. Écrire l'architecture avec les équations logiques pour chaque bit.
  5. Compiler et corriger les erreurs éventuelles.
- ◆ Étape 5 : Simulation
  - Simuler university program file.
  - Appliquer manuellement des valeurs binaires sur A, B, Cin.
  - Observer les résultats sur S et Cout.
- ◆ 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...).
- ◆ Étape 7 : Programmer
  1. Dans la fenêtre Programmer, cocher **Program/Configure**.
  2. Cliquer sur **Start**.
  3. 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] |



