



TP 04 : Simulation et implémentation d'un programme VHDL pour un afficheur 7 segment

Thème : Compteur, diviseur d'horloge, 1 et 2 digits

Objectifs pédagogiques

- Comprendre le principe de fonctionnement d'un afficheur 7 segments.
- Concevoir en VHDL un décodeur BCD–7 segments.
- Simuler l'affichage des chiffres de 0 à 9 sur un seul afficheur.

Partie théorique

◆ L'afficheur 7 segments

- Un afficheur 7 segments est constitué de 7 LED disposées en forme de “8”, nommées a, b, c, d, e, f, g.
- Chaque chiffre est représenté par une combinaison particulière de segments allumés.
- Exemple :
Chiffre 0 → segments a, b, c, d, e, f allumés, g éteint.
Chiffre 1 → segments b, c allumés.
Chiffre 8 → tous les segments allumés.
- Afficheur à anode commun actif quand anode et a l'état haut et le segment à l'état bas.
- Afficher à cathode commun actif quand l'anode et a l'état bas et se segment à l'état haut.

◆ Codage BCD

- Chaque chiffre décimal (0–9) est représenté par un mot binaire de 4 bits (Binary Coded Decimal).
- Exemple :
0 → 0000
1 → 0001
...
9 → 1001

◆ Décodeur BCD–7 segments

- Entrée : un mot BCD (4 bits).
- Sortie : un vecteur de 7 bits (a..g).
- Fonction : associer chaque code BCD à la combinaison de segments correspondante.

Travail demandé

- ◆ Étape 1 : Définition de l'entité
 - Créer une entité nommée par exemple `bcd_to_7seg`.
 - Entrée : `bcd` (4 bits).
 - Sortie : `seg` (7 bits), `en0`.
- ◆ Étape 2 : Architecture
 - Décrire un process avec une instruction case qui associe chaque valeur BCD (0–9) à la combinaison de segments correspondante.
 - L'activation de l'anode commune du digit utilisé (`en0`).
- ◆ Étape 3 : Vérification
 - Vérifier que :
 - Pour `bcd = 0000` → sortie correspond au chiffre 0.
 - Pour `bcd = 0001` → sortie correspond au chiffre 1.
 - ...
 - Pour `bcd = 1001` → sortie correspond au chiffre 9.

Compteur 0–99 avec affichage multiplexé sur 2 digits

Objectifs pédagogiques

- Comprendre le codage BCD et le décodage vers 7 segments.
- Concevoir en VHDL un décodeur BCD–7 segments.
- Réaliser un compteur modulo 100 (0–99).
- Simuler le fonctionnement d'un affichage multiplexé sur deux digits 7 segments.

Partie théorique

- ◆ Comptage
 - Supposons que le compteur incrémente de 0 à 99.
 - Chaque valeur est un entier (ex. 0, 1, 2, ..., 99).
 - Pour afficher ce nombre sur deux digits (unités et dizaines), il faut séparer la valeur en deux parties :
 - Unités : $\text{valeur} \bmod 10$ → donne les chiffres des unités.
 - Dizaines : $\text{valeur} \div 10$ → donne les chiffres des dizaines.
- ◆ Multiplexage
 - Les deux digits partagent les 7 lignes de segments (a..g).
 - Chaque digit possède une anode de sélection.
 - Pour afficher deux chiffres différents :
 - Activer l'anode du premier digit et envoyer son code BCD (anode deuxième digit désactivé).
 - Puis activer l'anode du second digit et envoyer son code BCD (anode du premier digit désactivé).

- En répétant rapidement (fréquence de multiplexage > 100 Hz), l'œil humain perçoit les deux chiffres affichés simultanément.
- En simulation, on observe l'alternance des anodes et des segments.

Travail demandé

◆ Étape 1 : Fonction global

- Créer une entité counter_display:

Entrées :

- clk (horloge FPGA).
- Reset (remise à zéro).

Sorties :

- seg (vecteur des 7 segments partagés).
- an (vecteur des deux anodes, une pour les unités, une pour les dizaines).

◆ Étape 2 : Compteur modulo 10

- **Compteur modulo 10** : incrémente de 0 à 99 puis revient à 0. **Séparation en unités et dizaines.**
- Process fonctionne avec l'horloge de 1 Hz. (**Crée un process spécial de division d'horloge**).
- Signal: unit, dizaine: std_logic_vector (3 downto 0).

◆ Étape 3 : Décodeur BCD–7 segments

- Associer chaque chiffre (0–9) à la combinaison de segments correspondante.
- Utilisation de la condition case sur un signal BCD.
- Sorties: seg: std_logic_vector (6 downto 0).

◆ Étape 4 : Multiplexeur d'affichage

- Les segments a..g sont communs aux deux digits.
- Les anodes an (0) et an (1) sélectionnent respectivement le digit des unités et celui des dizaines.
- Alternier rapidement entre les deux digits en activant l'anode correspondante et en envoyant le code segments avec un signal alterne. En créant une horloge d'alternance de 1 kHz (**Crée un Process spéciale division d'horloge**).
- En simulation, on observe l'alternance rapide :
Quand le signal alterne et a l'état haut, an (0) est actif et an (1) désactivé → décodage des unités par BCD.
Quand le signal alterne et a l'état bas, an (1) est actif an (0) désactivé → décodage des dizaines par BCD.
- Process en fonction des signaux : unit, dizaine. Alterne (ou affectation conditionnelle).
- Sorties: an: std_logic_vector (1 downto 0).

◆ Étape 5 : Simulation

- Simuler le projet complet (compteur + multiplexeur + décodeur).

- Observer :
 Les sorties seg changent selon le chiffre affiché.
 Les sorties an alternent entre 10 et 01.
 Vérifier que la séquence affichée correspond bien au comptage de 0 à 99.

ANNEX

#-----GLOBAL-----#

set_location_assignment	PIN_12	-to	clk
set_location_assignment	PIN_41	-to	rst_n

#-----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]

#-----DIG-----#

set_location_assignment	PIN_87	-to	dig[0]
set_location_assignment	PIN_86	-to	dig[1]
set_location_assignment	PIN_85	-to	dig[2]
set_location_assignment	PIN_84	-to	dig[3]
set_location_assignment	PIN_95	-to	dig[4]
set_location_assignment	PIN_92	-to	dig[5]
set_location_assignment	PIN_91	-to	dig[6]
set_location_assignment	PIN_89	-to	dig[7]

set_location_assignment	PIN_76	-to	seg[0]
set_location_assignment	PIN_83	-to	seg[1]
set_location_assignment	PIN_81	-to	seg[2]
set_location_assignment	PIN_75	-to	seg[3]
set_location_assignment	PIN_74	-to	seg[4]
set_location_assignment	PIN_78	-to	seg[5]
set_location_assignment	PIN_82	-to	seg[6]
set_location_assignment	PIN_77	-to	seg[7]

