



TP 03 : Initiation aux instructions séquentielles

Commande servomoteur

Thème : Compteur, diviseur d'horloge, PWM pour servomoteur

Objectifs pédagogiques

- Comprendre le rôle des **instructions séquentielles** en VHDL et leur différence avec les instructions concurrentes.
- Apprendre à produire un signal carré à fréquence réduite.
- Savoir utiliser un **process** avec une **liste de sensibilité** et un **front d'horloge** pour décrire des comportements synchrones.
- Apprendre à générer une **horloge dérivée** à partir d'une horloge maître de **50 MHz**.
- Savoir **modifier le rapport cyclique (duty cycle)** d'un signal périodique.
- Être capable d'écrire un **code VHDL séquentiel** permettant de générer plusieurs horloges de fréquences différentes.

Partie théorique

- ◆ 1. Instructions séquentielles
 - Contrairement aux instructions concurrentes (qui s'exécutent en parallèle), les instructions séquentielles s'exécutent **dans l'ordre d'écriture** à l'intérieur d'un bloc process.
 - Elles permettent de décrire des comportements temporels : compteurs, registres, machines à états.
 - Exemples : if...then, case...when, for...loop, affectations séquentielles.
- ◆ 2. Le process en VHDL
 - Un process est un bloc séquentiel qui s'exécute lorsqu'un signal de sa **liste de sensibilité** change.
 - La **liste de sensibilité** contient les signaux surveillés. Exemple :

Process (clk, reset)

- Ici, le process est exécuté à chaque changement de clk ou de reset.
- Pour décrire un comportement synchrone, on teste le **front montant de l'horloge** :

```
Process (clk, reset)
begin
  if rising_edge (clk) then
    -- instructions séquentielles
  end if;
end process;
```

- Cela permet de modéliser le fonctionnement d'un registre ou d'un compteur cadencé par l'horloge.
- ♦ 3. Génération d'une horloge dérivée pour générer un signal carré
 - Horloge maître : **50 MHz** → période = 20 ns.
 - Pour obtenir une fréquence cible f_{out} : $N = \frac{f_{maître}}{f_{sortie}} = \frac{50\,000\,000}{f_{out}}$
 - Le compteur doit compter de **0 à N-1** inclus.
Exemple : pour 1 kHz → N = 50 000. Le compteur va de 0 à 49 999.
 - Pour un signal **carré** (rapport cyclique 50 %), on bascule la sortie à la moitié de la période :
Sortie = '1' pendant N/2 cycles
Sortie = '0' pendant N/2 cycles
 - On peut directement compter Jusqu'à la moitié (N/2)-1 À la fin du compteur on inverse la sortie.
Sortie = not (sortir).
- ♦ 4. Génération d'une horloge dérivée avec modification du rapport cyclique
 - Le rapport cyclique est la proportion de temps où le signal est à 1.
 - Pour un compteur allant de 0 à N-1 :
 - Si compteur < N_duty → sortie = 1
 - Sinon → sortie = 0
 - Exemple : pour un duty de 30 % à 1 kHz → N_duty = 50 000 * 0,3 = 15 000.

Étapes de codage VHDL pour générer un signal carré dérivée

- ♦ Déclarer l'entité :
 - Entrées : clk_50MHz, reset
 - Sortie : clk_out (signal carré) (**Buzzer**)
- ♦ Définir les constantes :
 - PERIOD = N (ex. 50 000 pour 1 kHz)
- ♦ Déclarer les signaux internes :
 - Signal compteur (counter : integer range 0 to PERIOD-1:=0)
 - Signal interne pour les sorties (clk_int : std_logic := '0')
- ♦ Écrire le process séquentiel :
 - Liste de sensibilité : (clk_50MHz, reset).
 - Au front montant de clk_50MHz :
 - Incrémenter le compteur
 - Si compteur = N-1 → remise à zéro
 - Si compteur < N/2 → clk_int <= '1'
 - Sinon → clk_int <= '0'
 - Affecter la sortie :
clk_out <= clk_int

- ◆ Générer plusieurs horloges
 - Répéter la logique pour chaque fréquence souhaitée.
 - Exemple :
 - clk_1kHz basé sur un compteur de 50 000.
 - clk_100kHz basé sur un compteur de 500.
 - clk_10Hz basé sur un compteur de 5 000 000.
- ◆ Simulation
 - Créer un fichier de test .vwf dans Quartus II.
 - Appliquer un reset puis laisser tourner l'horloge maître.
 - Vérifier que les sorties générées correspondent aux fréquences attendues.
 - Vérifier que la modification du duty cycle change bien la largeur des impulsions.

Étapes de codage VHDL d'une horloge dérivée avec modification du rapport cyclique (PWM)

- ◆ Déclarer l'entité et des signaux :
 - Entrées : clk_50MHz, reset
 - duty: std_logic_vector (1 downto 0)
 - Sortie : clk_modulated (**Buzzer**)
 Définir les constantes :
 - PERIOD = N (ex. 50 000)
 Déclarer les signaux internes :
 - counter: integer range 0 to PERIOD-1
 - duty_int : unsigned (1 downto 0) ← conversion de duty en integer
 - clk_int : std_logic
- ◆ Écrire le process séquentiel :
 - Au front montant de clk_50MHz :
 - Incrémenter le compteur
 - Si compteur < N/(to_integer(duty_int)+1) → clk_int <= '1'
 - Sinon → clk_int <= '0'
 - Si compteur = N-1 → remise à zéro
- ◆ Affecter des signaux internes aux port d'entrée sortie :
 - clk_modulated <= clk_int
 - duty_in <= duty
- ◆ Simulation
 - Créer un fichier de test .vwf dans Quartus II.
 - Appliquer un reset puis laisser tourner l'horloge maître.
 - Vérifier que les sorties générées correspondent aux fréquences attendues.
 - Vérifier que la modification du duty cycle change bien la largeur des impulsions.

Commande d'un servomoteur avec FPGA et signal PWM

Objectifs pédagogiques

- Comprendre le principe de commande d'un **servomoteur** à l'aide d'un **FPGA**.
- Étudier la **caractérisation du signal PWM** nécessaire pour piloter l'angle de rotation.
- Concevoir en VHDL un générateur de PWM permettant de faire varier la position du servomoteur.
- Vérifier expérimentalement la correspondance entre le **rapport cyclique** et l'**angle de rotation**.

Partie théorique

♦ Servomoteur

- Un **servomoteur** est un moteur à courant continu muni d'un réducteur et d'un système de retour (potentiomètre + électronique interne).
- Il est commandé par un **signal PWM** appliqué sur son entrée de commande.
- L'angle de rotation est proportionnel à la **largeur de l'impulsion** du PWM.
- Fréquence du PWM pour servomoteur

Fréquence typique : 50 Hz

Période correspondante : 20 ms

Cette fréquence est un standard pour les servos RC. La période doit rester stable

♦ Caractérisation du signal PWM pour servomoteur

- **Fréquence typique** : 50 Hz (période = 20 ms).
- **Largeur d'impulsion** :
- 1 ms → angle minimum ($\approx 0^\circ$).
- 1,5 ms → position médiane ($\approx 90^\circ$).
- 2 ms → angle maximum ($\approx 180^\circ$).

 Ainsi, le **rapport cyclique** varie entre 5 % et 10 % pour couvrir toute la plage de rotation.

♦ Génération du PWM avec FPGA/CPLD

Période de 20 ms en cycle d'horloge :

$$N_{\text{periode}} = 50 \text{ MHz} \times 20 \text{ ms} = 1\,000\,000 \text{ cycles.}$$

Largeurs d'impulsion en cycles :

- 1 ms → $50 \text{ MHz} \times 1 \text{ ms} = 50\,000 \text{ cycles}$
- 1,5 ms → 75 000 cycles
- 2 ms → 100 000 cycles

Principe de génération :

- Compteur de période : compte de 0 à $N_{\text{periode}} - 1$ puis se réinitialise.
- Comparateur de largeur : la sortie PWM est à 1 tant que le compteur $< N_{\text{puls}}$, sinon 0.

- Consigne d'angle : convertie en $N_{impulsion}$ par une formule linéaire.

$$N_{puls} = 50000 + 50000 * \frac{\theta}{180}$$

👉 Rampe de position : changez l'angle progressivement (pas de 2–5° toutes les 20–40 ms) pour éviter des à-coups.

🧱 Étapes de conception en VHDL

♦ Étape 1 : Spécifications

- Entrées :
 - clk : horloge FPGA (ex. 50 MHz).
 - pos : consigne de position (codée sur quelques bits). **(Switch1..3)**
- Sortie :
 - pwm_out : signal PWM appliqué au servomoteur. (Buzzer ou LED 1)

♦ Étape 2 : Conception du générateur PWM

- Diviser l'horloge FPGA pour obtenir une période de 20 ms.
- Générer une impulsion dont la largeur varie entre 1 ms et 2 ms selon la consigne.
- Établir la **relation entre la largeur d'impulsion et l'angle de rotation**.
- Associer chaque valeur de pos à une largeur d'impulsion correspondante.

♦ Étape 3 : Simulation

- Vérifier que la largeur d'impulsion change correctement quand pos varie.
- Observer les chronogrammes du signal PWM.

♦ Étape 4 : Implémentation matérielle

- Programmer le FPGA avec le code VHDL.
- Connecter la sortie pwm_out à l'entrée de commande du servomoteur.
- Alimenter le servomoteur avec une source externe adaptée (5 V typiquement).
- Tester la variation de l'angle en changeant la valeur de pos.

♦ Étape 5 : PIN Planner

Associer les signaux logiques du projet VHDL (ex clk, pos, pwm_out) 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**.

ANNEX

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

```
set_location_assignment PIN_12      -to  clk
set_location_assignment PIN_41      -to  rst_n
```

#-----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-----#

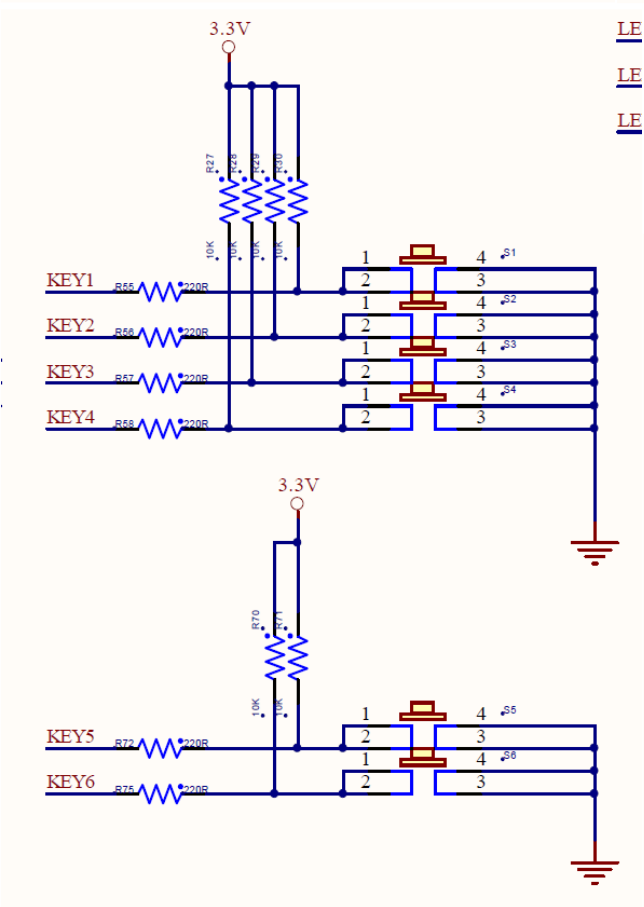
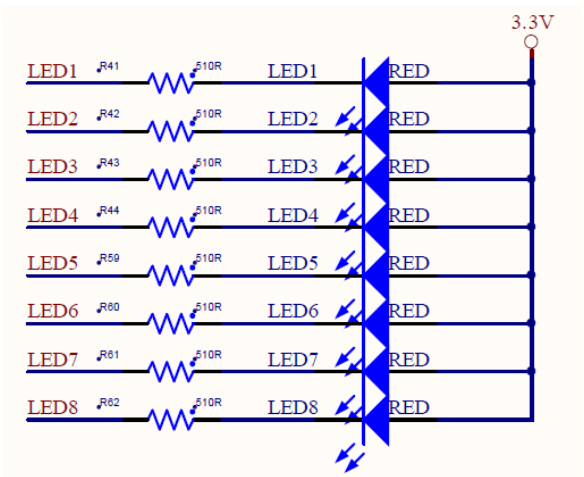
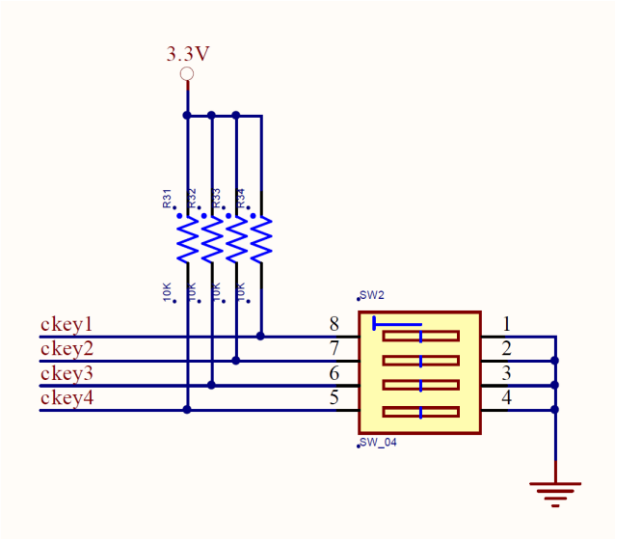
```
set_location_assignment PIN_49      -to  key[1]
set_location_assignment PIN_48      -to  key[2]
set_location_assignment PIN_47      -to  key[3]
set_location_assignment PIN_44      -to  key[4]
set_location_assignment PIN_43      -to  key[5]
set_location_assignment PIN_42      -to  key[6]
```

#-----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]
```

#-----Buzzer-----#

```
set_location_assignment PIN_73      -to  beep
```



LE
LE
LE

