TP1 : Compteur BCD, HeartBeat et Chenillard

1/ Compteur BCD affichage 7 segments (10 points)

Objectifs

- Implémenter un compteur BCD (logique synchrone) et un décodeur 7-segments en VHDL (logique combinatoire).
- Vérifier le compteur par simulation logique (ModelSim).
- Synthétiser le projet complet.
- Télécharger le projet sur la carte

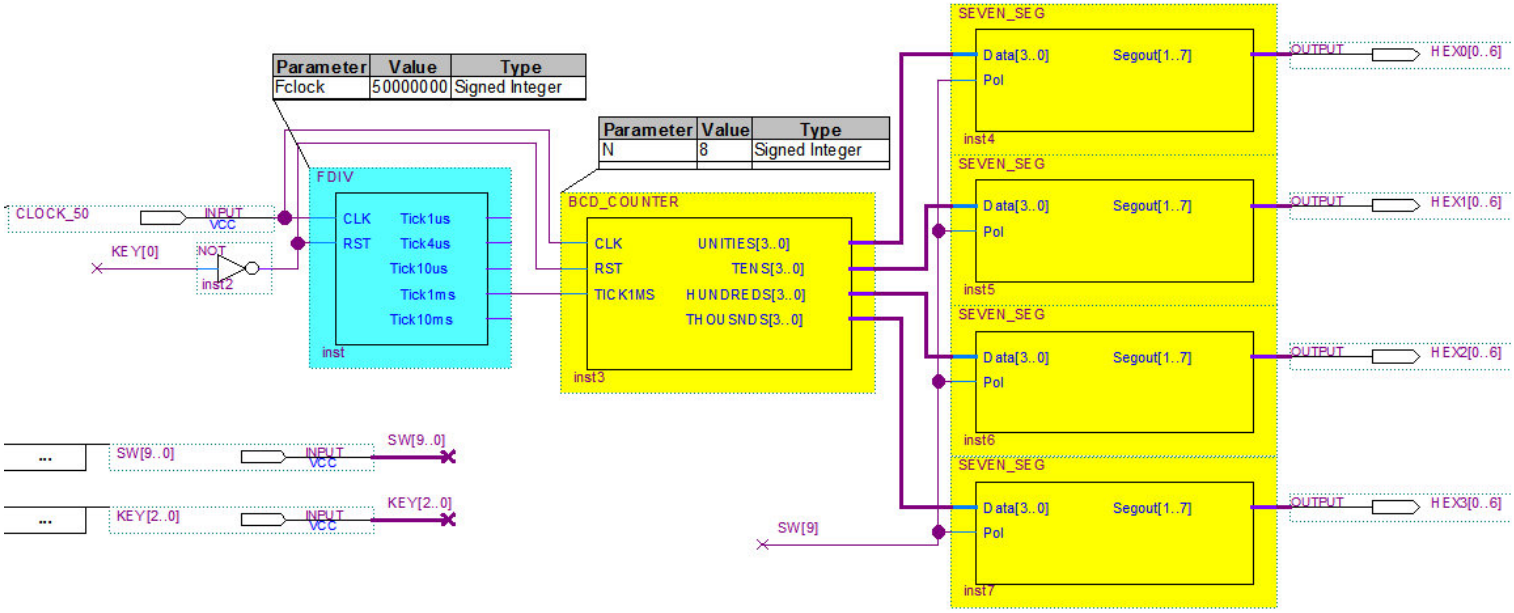
Cet exercice doit afficher un compteur de 0 à 9999 sur les afficheurs 7-segments de la carte. Seul le compteur et le décodeur sont à écrire, le niveau supérieur de cette application étant fourni. L'affichage et la génération des tops à 1 ms est pris en charge, et un banc de test unitaire est également fourni.

Malgré sa simplicité, cet exercice permet de mettre en oeuvre l'essentiel de la méthodologie et des outils de conception.

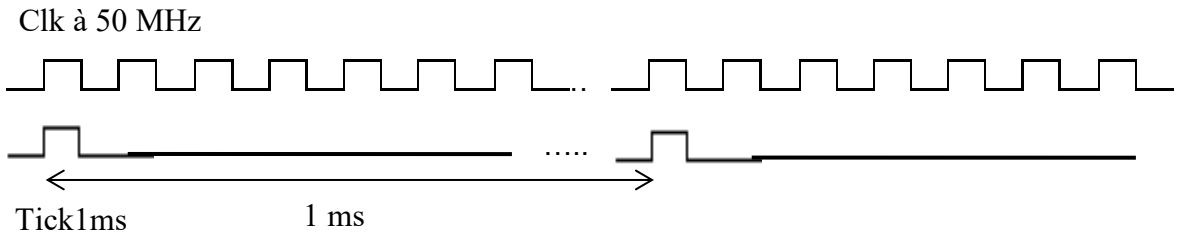
Description de l'exercice

L'horloge d'entrée du FPGA est une horloge de 50 MHz. On désire compter de 0000 à 9999 au rythme d'une incrémentation toutes **les « N » millisecondes**, N étant paramétrable à l'instanciation par un paramètre **« generic »** : **N**. Le compteur doit repasser à 0000 lorsque l'on se trouve à 9999.

Voici le schéma descriptif de la fonctionnalité demandée :



Le bloc FDIV fournit un signal **`Tick1ms`** actif (niveau logique '1') toutes les 1 millisecondes durant un cycle d'horloge à 50 MHz.



Le signal **Tick1ms** arrive sur un port d'entrée du bloc BCD_Counter et doit être utilisé pour réaliser l'incrémentation du compteur toutes les **N ms**.

Il est hors de question d'utiliser Tick1ms comme horloge (l'horloge est toujours Clk, l'horloge principale !!!) Il vous faudra réaliser un test sur l'état de Tick1ms.

Le compteur BCD a 4 ports de sortie : un pour les unités, un pour les dizaines, un pour les centaines et le dernier pour les milliers. Chacun de ces ports est bien sûr codé sur 4 bits, afin de pouvoir compter de 0 à 9 inclus ... (et pas au-delà).

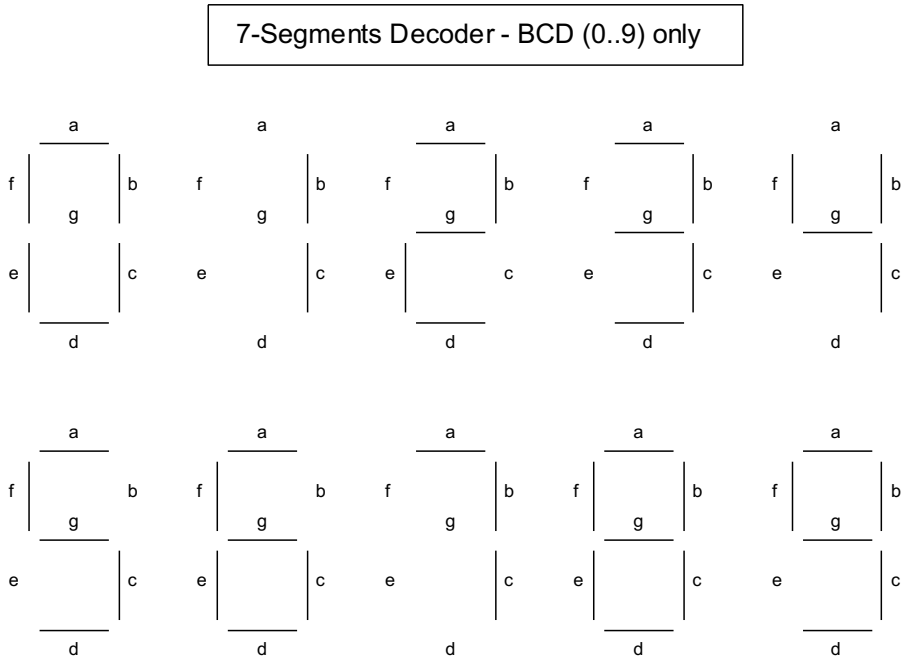
Ces ports sont ensuite encodés par le bloc SEVEN_SEG pour pouvoir être affichés sur les afficheurs 7 segments.

Décodeur 7-segments

Voici le fonctionnement d'un afficheur 7-segments :

Les segments sont traditionnellement identifiés par les lettres de A à G (ou « a » .. « g ») suivant la convention représentée. Dans notre module, la sortie est un vecteur croissant de 1 à 7 correspondants aux segments A à G.

De plus, nous avons une entrée de sélection de polarité « Pol » :
Pol = 1 signifie des afficheurs actifs niveau 1
Pol = 0 signifie des afficheurs actifs niveau 0
Ainsi, pour afficher le chiffre 1, on devra établir SegOut = 0110000 si Pol=1 et 1001111 si Pol=0.



Ce qu'il faut faire

Copier le dossier TP1_BCD_HeartBeat.zip

Ce répertoire est sectionné en 3 sous répertoires qui séparent bien les trois étapes nécessaires pour la réalisation complète d'un design (édition, simulation et synthèse) :

- **src** : qui contient les fichiers sources VHDL nécessaires pour le design (dont les fichiers à compléter).
- **simu** : qui contient les fichiers nécessaires à la simulation (Test Bench, script de simulation, modèle VHDL, etc.…)
- **fit :** qui contient les fichiers nécessaires à la synthèse (Script de synthèse, rapport de synthèse, placement / routage, analyse des Timing, etc…)
-

Pour l'ensemble du TP1, vous pouvez utiliser ***NotePad ++*** ou ***VSCode*** comme éditeur de VHDL et ***Modelsim*** pour la simulation. Un ensemble de scripts sont fournis et permettent de ne pas perdre trop de temps pour la simulation.

Écrire le compteur BCD en VHDL

- ➢ Ouvrez le fichier **bcd_counter.vhd** sous le répertoire **TP1_BCD_HeartBeat/src**. Ce fichier contient déjà la déclaration des ports de l'entité BCD_COUNTER, et également la déclaration de l'architecture de l'entité ainsi que le modèle de process synchrone avec reset asynchrone. Vous devez insérer votre code là où les commentaires l'indiquent, et pas ailleurs. Ceci permet d'obtenir un code qui respecte les règles usuelles de codage, de synthèse, et de qualité.
- ➢ Codez un compteur BCD qui compte **toutes les N x 1 ms** de 0000 à 9999. N'oubliez pas que le compteur doit repasser à 0000 lorsque l'on est à la valeur 9999...

Vérifier le compteur (par simulation RTL) (3 points)

- ➢ Lancer ModelSim.
- ➢ Dans Modelsim, aller sous le répertoire **TP1_BCD_HeartBeat/simu** (File > Change Directory ou commande `cd`).
- ➢ Taper la commande : "`do simu.do`" qui lance le script de compilation et de simulation.
- ➢ Si il y a une erreur de compilation, analyser la nature de l'erreur, corriger et recommencer.
- ➢ Vérifier dans le transcript le succès de la simulation. En effet, le banc de test fourni est auto-vérifiant : il s'assure rapidement par des tests simples que le compteur semble fonctionner. C'est aussi une bonne idée de regarder dans le visualisateur des waveforms.

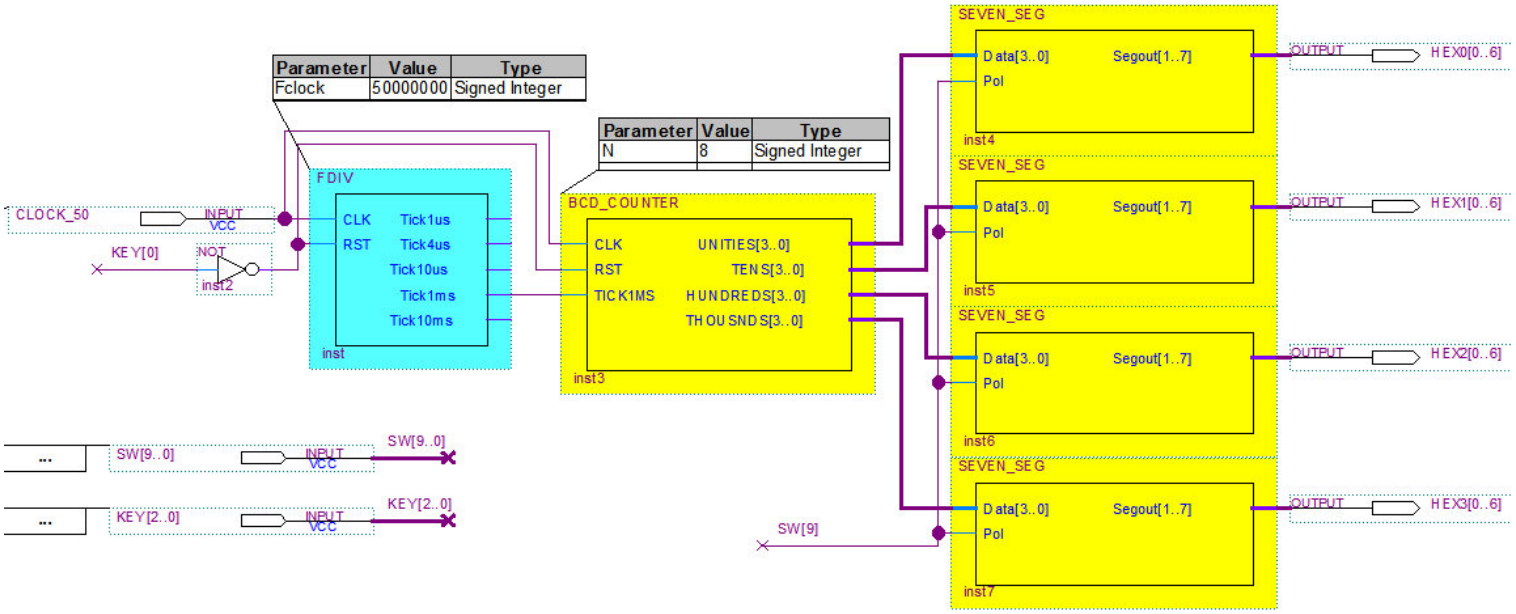
Écrire le décodeur 7-segments

- ➢ Ouvrir le fichier `Seven_Seg.vhd`.
- ➢ Écrire le décodage en tenant compte des explications de l'énoncé (ne pas oublier la gestion de la polarité). Nous n'avons pas fourni de banc de test pour cette fonction car si vous deviez l'écrire, vous écririez certainement la même table dans le banc de test que dans le décodeur, ce qui n'a (pratiquement) aucune valeur au titre de vérification ! Il faut accepter dans ce cas la vérification visuelle du résultat.
- ➢ Noter que l'on ne précise rien pour les codes 10..15 (décimal), écartés par l'énoncé. Dans ce cas, il peut être intéressant d'optimiser la logique en laissant la synthèse adopter

un code optimal pour ces cas non demandés (aide : il existe une des neufs valeurs `std_logic` qui est réservée à cet effet).

Écrire le module top level

- ➢ Ouvrir le fichier `TP1_BCD.vhd`.
- ➢ Compléter le module TP1_BCD qui permet d'instancier le composant FDIV, le composant BCD_COUNTER et 4 fois le module SEVEN_SEG. Comme sur le schéma ci-dessous :



Synthétiser et charger le composant programmable sur la maquette.

Créez un projet sur Quartus dans le répertoire **fit**, nommez le **TP1_BCD** sélectionnez le FPGA relatif à la carte de développement que vous utilisez (voir le « User Manuel » de cette carte). Rajouter vos sources en sélectionnant les sources qui sont dans le répertoire **src** (sans les copier dans le répertoire fit).

Sélectionnez le composant **TP1_BCD** comme entité de plus haute hiérarchie (top-level). Pour cela, sélectionnez le « File » **TP1_BCD.vhd** dans le Project Navigator, faites un clic droit avec la souris et sélectionnez « Set as top-level Entity ».

Maintenant il faut faire l'assignement des pins d'entrées/sorties :

Lancer une première fois la synthèse de votre projet (**Processing -> Start compilation**). Ensuite assignez les pins du FPGA depuis le menu **Assignements → Pin Planner** en vous reportant au manuel de la carte pour choisir les bons numéros de pin sur la colonne **Location.**

Après l'assignement des pins, il faut relancer la synthèse de votre projet.

Programmation du FPGA :

- ➢ Ouvrez le Programmateur qui se trouve dans **Tools → Programmer**.
- ➢ Connecter la carte de développement avec le cable USB (USB Blaster)

- En haut à gauche de cette fenêtre, en face de Hardware Setup, si il n'y a pas noté USB_Blaster, cliquez sur Hardware Setup et choisissez USB_Blaster et cliquez sur Close.
- Lorsque USB_Blaster est bien sélectionné, cliquez sur Start et normalement le bitstream (fichier .sof) se charge dans le FPGA.
- La carte doit alors afficher le compteur rapide. Vérifier que l'affichage est correct, sinon corriger le module de décodage 7-segments et recommencer.

Vérification du travail réalisé (3 points)

Vous devrez faire vérifier le fonctionnement du compteur BCD à l'enseignant encadrant. Ensuite, vous noterez votre nom au tableau en face duquel vous noterez le nombre de Logic Element (LE) utilisé pour le réaliser. Relever également la fréquence maximum d'utilisation

Le nombre de LE et la fréquence max sont précisés dans le rapport de synthèse édité par Quartus.

Amélioration 1 : (2 points)

- Pour l'instant, la vitesse d'incrémentation est fixée par un paramètre générique (N). Il faut recompiler le design à chaque fois que l'on veut modifier cette valeur.
- Dans cette partie, on propose de pouvoir régler la vitesse d'incrémentation du compteur à partir des interrupteurs (Switch, SW). Ainsi, on pourra régler une vitesse d'incrémentation entre 1 et 256 ms. Il faudra incrémenter le compteur après chaque « valeur entière de SW(7 downto 0) + 1 ».
- Pour éviter de toucher au code de la partie précédente, on propose de créer une nouvelle entité pour le module BCD_COUNTER (il faudra mettre la première entité/architecture en commentaire)

Amélioration 2 : (2 points)

- Pour l'instant, la vitesse d'incrémentation est fixée par un paramètre générique (N) ou par la valeur des interrupteurs (SWITCH ou SW).
- Dans cette partie, on propose d'incrémenter le compteur BCD avec l'aide d'un bouton poussoir (KEY). Cela consiste donc à faire un compteur d'impulsion. Chaque impulsion (chaque appui sur le bouton) augmente le compteur, et la valeur décimale du compteur est affichée en temps réel sur les afficheurs 7 segments.
- Attention, il est interdit d'utiliser l'instruction rising_edge() sur un autre signal que l'horloge (clk). Il faut utiliser un circuit de détection de front.
- Pour éviter de toucher au code des parties précédentes, on propose de créer une nouvelle entité pour le module BCD_COUNTER (il faudra mettre les premières entité/architecture en commentaire)

Le travail présenté doit être uniquement le travail de l'étudiant. Présenter le travail d'un autre sera sévèrement pénalisé.

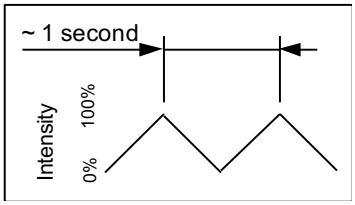
2/ PWM et LED qui bat (HeartBeat) (4 points)

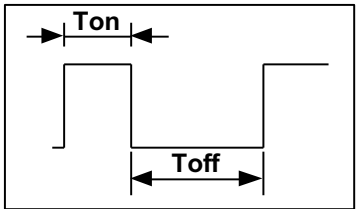
Objectifs

- ➢ Comprendre la modulation PWM.
- ➢ Implémenter de la logique synchrone fonctionnant à différents rythmes,
- ➢ Construire une séquence simple (comptage/décomptage).
- ➢ Construire le niveau supérieur de la hiérarchie (*Top Level*).

Le but de cet exercice est d'afficher sur une des LEDs de la carte un « coeur qui bat» (*HeartBeat* en anglais). Pour cela, il faut moduler l'intensité perçue de la diode luminescente par une fonction triangulaire comme ci-contre. Il n'est pas demandé une période très précise, mais une variation « autour » de 1 hertz (entre 0.5 et 2 Hz par exemple).

Pour cet effet de variation proportionnelle, nous utiliserons une technique dite Modulation de Largeur d'Impulsion, soit PWM en anglo-saxon (Pulse Width Modulation) représentée ci-desous.





L'intensité moyenne est donc proportionnelle au rapport cyclique **Ton / (Ton + Toff).** On peut donc faire varier **Ton** en gardant **Ton+Toff** constant. Un codage sur 8 bits est plus que suffisant (5 bits auraient suffit pour cette application), et permettra de réutiliser ce module PWM à d'autres fins.

Remarque : dans la pratique, l'intensité lumineuse perçue n'est pas une fonction linéaire, mais nous négligerons ceci pour cette application simple.

Pour que l'œil ne voit pas les allumages et extinctions de la LED, il faut que la période **Ton+Toff** soit suffisamment faible. Une fréquence d'au moins 50 Hz (Ton+Toff < 20 ms) suffit. D'un autre côté, on évitera également les périodes trop courtes peu compatibles avec les temps de commutation de l'électronique de commande, et génératrices d'interférences. Un bon choix se situe entre 60 Hz (16.7 ms) et 400 Hz (2.5 ms).

En divisant la période choisie en 2^N parties avec N=8 (soit 256), on arrive à une unité d'au moins 2.5 ms /256 soit 10 us. Pour la modulation triangulaire, avec la même précision de 8 bits pour le compteur / décompteur binaire, on arrive pour un cycle de 1 seconde à un rythme de 1/ 512 = 2 ms, qui est voisin de la durée d'un cycle PWM. Une solution possible consiste donc à construire un compteur 8 bits pour le PWM qui s'incrémente toutes les 10 µs, et à l'issue de chaque cycle PWM, on incrémente ou décrémente un autre compteur 8 bits qui représente le rapport cyclique. Le cycle total durera 512 * 256 * 10µs = ~1,33 seconde, ce qui est conforme au cahier des charges.

Attention : il est exigé d'avoir une croissance suivie d'une décroissance, sans saut brusque.

Ce qu'il faut faire

Coder et Vérifier le Module HeartBeat.vhd

- ➢ **Coder** en un ou en deux process synchrones la génération de PWM et la rampe de consigne.
- ➢ **Simuler** grâce au banc de test fourni, qui n'est PAS auto-vérifiant !
  Il faudra regarder la forme d'onde.
  On remarquera que le banc de test définit une horloge système lente afin d'accélérer la simulation (0 à 1.5 seconde de temps réel).

NB : la solution proposée représente une quarantaine de lignes de code.

Coder le Niveau Supérieur : TP1_HeartBeat.vhd

- ➢ Dans le fichier **`TP1_HeartBeat.vhd`**, instancier et interconnecter les modules **FDIV** et **HeartBeat**.
  La déclaration d'entité est déjà faite.
  On pilotera la sortie nommée LED(0), qui est active niveau haut (mais ceci n'a pas d'importance dans cette application).

On pourrait imaginer vérifier ce niveau par simulation, mais si les deux modules qui le constituent ont été correctement vérifiés, ceci n'est pas indispensable, d'autant que la simulation serait assez longue.

Nous allons donc passer au test sur la carte

Synthétiser et charger le composant programmable sur la maquette.

Reprenez le projet Quartus précédemment créer dans le répertoire **fit**. Rajouter vos sources en sélectionnant les sources qui sont dans le répertoire **src** (sans les copier dans le répertoire fit).

Sélectionnez le composant **TP1_HeartBeat** comme entité de plus haute hiérarchie (top-level). Pour cela, sélectionnez le « File » **TP1_HeartBeat.vhd** dans le Project Navigator, faites un clic droit avec la souris et sélectionnez « Set as top-level Entity ».

Maintenant il faut faire l'assignement des pins d'entrées/sorties :

Lancer une première fois la synthèse de votre projet (**Processing -> Start compilation**). Ensuite assignez les pins du FPGA depuis le menu **Assignements → Pin Planner** en vous reportant au manuel de la carte pour choisir les bons numéros de pin sur la colonne **Location.**

Après l'assignement des pins, il faut relancer la synthèse de votre projet.

Programmation du FPGA :

- ➢ Ouvrez le Programmateur qui se trouve dans **Tools → Programmer**.
- ➢ Connecter la carte de développement avec le cable USB (USB Blaster)
- ➢ En haut à gauche de cette fenêtre, en face de Hardware Setup, si il n'y a pas noté USB_Blaster, cliquez sur Hardware Setup et choisissez USB_Blaster et cliquez sur Close.
- ➢ Lorsque USB_Blaster est bien sélectionné, cliquez sur Start et normalement le bitstream (fichier .sof) se charge dans le FPGA.

Vérification du travail réalisé :

Vous devrez faire vérifier le fonctionnement du HeartBeat à l'enseignant encadrant. Ensuite, vous noterez votre nom au tableau en face duquel vous noterez le nombre de Logic Element (LE) utilisé pour le réaliser. Relever également la fréquence maximum d'utilisation

Le nombre de LE et la fréquence max sont précisés dans le rapport de synthèse édité par Quartus.

3/ Chenillard sur 10 LEDs (LED Chaser) (3 points)

Objectif :

L'objectif de ce projet est de concevoir un **jeu de lumière** (chenillard) en VHDL, où une série de **10 LEDs** s'allume successivement. L'utilisateur pourra ajuster la **vitesse de défilement** des LEDs à l'aide d'un **interrupteur unique**. Le chenillard pourra également être **démarré** ou **arrêté** à l'aide d'un bouton poussoir.

Spécifications Techniques :

- **Entrées** :
  - **1 interrupteur** pour régler la vitesse du chenillard. La vitesse sera sélectionnée en fonction de l'état de cet interrupteur, qui alternera entre **deux vitesses** : **rapide** ou **lente**.
  - Un **bouton poussoir** pour **démarrer/arrêter** le chenillard.
  - **Horloge** (fournie par la carte FPGA).
  - **Signal de reset** (pour réinitialiser le système à la position initiale).
- **Sorties** :
  - **10 LEDs** pour afficher la séquence du chenillard.
    - Une seule LED sera allumée à la fois, et celle-ci se déplacera de gauche à droite, puis de droite à gauche, créant un effet de balayage.

Comportement attendu :

1. **Initialisation** :
   - Le chenillard commence avec une seule LED allumée à l'extrémité gauche (LED0).
   - Si le **reset** est activé, le chenillard revient à cette position initiale (LED0 allumée, autres éteintes).
2. **Déroulement du chenillard** :
   - Le chenillard commence dès que le bouton **démarrage** est pressé.
   - Une seule LED est allumée à un instant donné, et elle se déplace de gauche à droite sur les **10 LEDs** (LED0 → LED1 → … → LED9), puis de droite à gauche (LED9 → LED8 → … → LED0), créant un effet de balayage.
3. **Contrôle de la vitesse** :
   - **Un interrupteur unique** contrôle la **vitesse** du chenillard :
     - **OFF** (0) : Chenillard **lent**.
     - **ON** (1) : Chenillard **rapide**.
4. **Démarrage/Arrêt du chenillard** :

- Le bouton poussoir permet de **démarrer** et **arrêter** le chenillard. Lorsqu'on appuie une première fois sur le bouton, le chenillard démarre. Un nouvel appui stoppe le chenillard à la position actuelle.
- Lorsque le chenillard est arrêté, la LED allumée reste à sa position jusqu'à la reprise du mouvement.

5. **Anti-rebond (Debouncing)** :
   - La détection des appuis sur le bouton poussoir sera plus fiable avec un système d'**anti-rebond** pour éviter la détection de multiples pressions en raison de fluctuations électriques.

Contraintes matérielles :

- Utilisation d'une **carte FPGA** avec :
  - **10 LEDs** pour visualiser le chenillard.
  - **1 bouton poussoir** pour démarrer/arrêter le chenillard.
  - **1 interrupteur** pour contrôler la vitesse (lent ou rapide).
  - Un signal de **reset** pour réinitialiser le système.

Diagramme des Entrées/Sorties :

Signal	Type	Description
clk	Entrée	Horloge du FPGA
reset	Entrée	Reset pour réinitialiser le chenillard
btn_start	Entrée	Bouton poussoir pour démarrer/arrêter le chenillard
switch_speed	Entrée	Interrupteur pour régler la vitesse (lent ou rapide)
leds[9:0]	Sortie	LEDs pour afficher le chenillard

Fonctionnement interne :

1. **Déplacement des LEDs** :
   - Utilisation d'un **compteur** pour représenter la position de la LED actuellement allumée.
   - Le compteur est incrémenté à chaque cycle du chenillard pour déplacer la LED de gauche à droite.
   - Lorsque la LED atteint l'extrémité droite (LED9), le sens du chenillard est inversé et le compteur est décrémenté pour revenir vers la gauche.
2. **Vitesse réglable** :
   - L'interrupteur unique contrôle la **vitesse de défilement** du chenillard.
   - **OFF** : Le chenillard défile **lentement** (grand délai entre chaque changement de LED).
   - **ON** : Le chenillard défile **rapidement** (délai réduit entre chaque changement de LED).
3. **Démarrage/Arrêt du chenillard** :
   - Lorsque le bouton poussoir est pressé, l'état du chenillard est inversé (si en marche, il s'arrête ; si arrêté, il redémarre).
4. **Anti-rebond pour le bouton poussoir** (optionnel)

- Un système d'anti-rebond filtre les fluctuations pour assurer que chaque pression du bouton poussoir est comptabilisée une seule fois.

Détail de l'algorithme :

1. **Initialisation** :
   - Le chenillard démarre avec la première LED allumée (LED0).
2. **Boucle principale** (processus sensible à l'horloge et au reset) :
   - Si le **reset** est activé, le chenillard revient à l'état initial (LED0 allumée, direction vers la droite).
   - Si le **bouton poussoir** est appuyé :
     - Si le chenillard est en marche, il s'arrête.
     - Si le chenillard est arrêté, il redémarre.
3. **Gestion de la vitesse** :
   - Si l'interrupteur est en position **OFF**, le délai entre chaque mouvement de LED est long (chenillard lent).
   - Si l'interrupteur est en position **ON**, le délai entre chaque mouvement de LED est court (chenillard rapide).
4. **Changement de direction** :
   - Lorsque la LED atteint **LED9** (à droite), le sens du chenillard est inversé et la LED commence à se déplacer vers la gauche (LED8 → LED7 → ... → LED0).
   - Lorsque la LED atteint **LED0** (à gauche), elle repart vers la droite (LED1 → LED2 → ... → LED9)

Pour cette partie vous allez devoir écrire l'ensemble des codes nécessaires pour la synthèse et la vérification. Aucune source n'est fournie pour cet exercice. Vous pouvez vous aider des codes des exercices précédent pour réaliser le Top-Level, le banc de test et le script de simulation.

N'oubliez pas de bien organiser vos codes dans les répertoires SRC, SIMU et FIT.

Vérification du travail réalisé :

Vous devrez faire vérifier le fonctionnement du Chenillard à l'enseignant encadrant. Ensuite, vous noterez votre nom au tableau en face duquel vous noterez le nombre de Logic Element (LE) utilisé pour le réaliser. Relever également la fréquence maximum d'utilisation

4/ Contrôleur de Volume Audio avec LED Bargraph (2 points)

Objectif :

L'objectif de cet exercice est de concevoir un **contrôleur de volume audio** en VHDL, qui affiche le niveau de volume sur un **bargraph LED** composé de **10 LEDs**. Le niveau de volume peut être ajusté à l'aide de **boutons poussoirs** pour **augmenter** ou **diminuer** le volume. Le système doit inclure une gestion d'anti-rebond pour les boutons et un indicateur de volume visuel sur les LEDs.

Spécifications Techniques :

- **Entrées** :
  - **Deux boutons poussoirs** :
    - Un bouton pour **augmenter** le volume.
    - Un bouton pour **diminuer** le volume.
  - **Horloge** (fournie par la carte FPGA).
  - **Signal de reset** pour réinitialiser le niveau de volume à zéro.
- **Sorties** :
  - **10 LEDs** pour afficher le niveau de volume sous forme de bargraph.
    - Les LEDs allumées représentent le niveau de volume actuel.
    - Par exemple, si 3 LEDs sont allumées, cela signifie que le volume est réglé à 30 %.

Comportement attendu :

1. **Initialisation** :
   - Au démarrage ou en cas de reset, le niveau de volume est réglé à **40%** (4 LEDs allumées).
   - Si le **reset** est activé, le volume revient à 40 %
2. **Ajustement du volume** :
   - Le niveau de volume peut être **augmenté** ou **diminué** par paliers avec des boutons poussoirs.
   - Chaque palier correspond à une **LED supplémentaire** allumée (sur 10 au total).
   - Si on appuie sur le bouton pour **augmenter** le volume, une LED supplémentaire s'allume jusqu'à ce que toutes les LEDs soient allumées (volume maximal).
   - Si on appuie sur le bouton pour **diminuer** le volume, une LED s'éteint jusqu'à ce que toutes les LEDs soient éteintes (volume minimum).
   - **Le niveau de volume doit rester dans la plage [0, 9]** correspondant aux 10 LEDs (volume de 0 à 100%).
3. **Anti-rebond (Debouncing)** :
   - Les boutons doivent inclure une gestion d'**anti-rebond** pour éviter les activations multiples involontaires lors d'un appui simple.

Contraintes matérielles :

- Utilisation d'une **carte FPGA** avec :
  - **10 LEDs** pour afficher le niveau de volume.
  - **2 boutons poussoirs** pour **augmenter** ou **diminuer** le volume.
  - Un signal de **reset** pour réinitialiser le niveau de volume.

Diagramme des Entrées/Sorties :

Signal	Type	Description
clk	Entrée	Horloge du FPGA
reset	Entrée	Reset pour réinitialiser le volume à 40%
btn_up	Entrée	Bouton pour augmenter le volume
btn_down	Entrée	Bouton pour diminuer le volume
leds[9:0]	Sortie	LEDs pour afficher le niveau de volume (bargraph)

Fonctionnement interne :

1. **Initialisation** :
   - Le niveau de volume commence à **40%** (4 LED allumée).
2. **Ajustement du volume** :
   - Lorsqu'un bouton est pressé, le volume est ajusté en **incrémentant** ou **décrémentant** la valeur du compteur qui représente le niveau de volume.
   - Le compteur peut varier entre **0 et 9**, correspondant aux 10 LEDs.
3. **Gestion des LEDs** :
   - Les LEDs sont gérées de manière à afficher le niveau de volume. Par exemple, si le volume est à 60%, les **6 premières LEDs** sont allumées.
   - Si le volume est à 0, **toutes les LEDs sont éteintes**. Si le volume est à 9, **toutes les LEDs sont allumées**.
4. **Anti-rebond pour les boutons** :
   - Un système d'anti-rebond filtre les fluctuations électriques pour assurer que chaque pression sur un bouton n'est comptabilisée qu'une seule fois.

Détail de l'algorithme :

1. **Initialisation** :
   - Le volume est initialisé à **4** (4 LEDs allumées).
   - Si le **reset** est activé, le volume revient à **4**
2. **Boucle principale** (processus sensible à l'horloge et au reset) :
   - Si le bouton **btn_up** est pressé, le volume est incrémenté (si le volume n'est pas déjà au maximum).
   - Si le bouton **btn_down** est pressé, le volume est décrémenté (si le volume n'est pas déjà au minimum).
3. **Affichage du volume sur les LEDs** :
   - Les LEDs sont mises à jour en fonction de la valeur du compteur de volume. Le nombre de LEDs allumées correspond au niveau actuel de volume.