

# TP1 : Compteur BCD, HeartBeat et Chenillard

## 1/ Compteur BCD affichage 7 segments

### 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

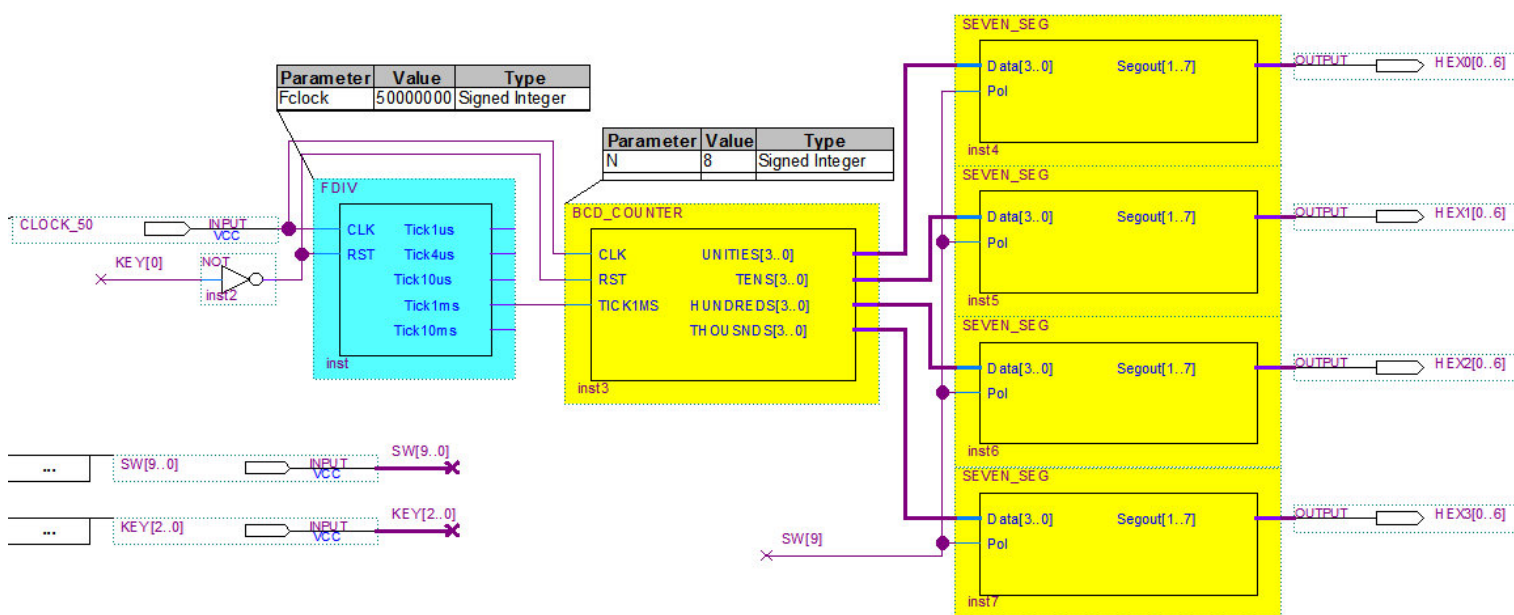
Cet exercice doit afficher un compteur de 0 à 9999 sur les afficheurs 7-segments de la carte. Seul le compteur et le décodeur sont à écrire, le niveau supérieur de cette application étant fourni. L'affichage et la génération des tops à 1 ms est pris en charge, et un banc de test unitaire est également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

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

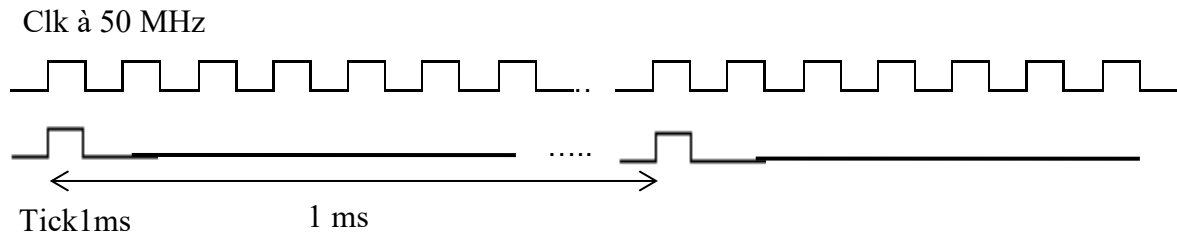
### Description de l'exercice

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'incrément 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 correspondant aux segments A à G.

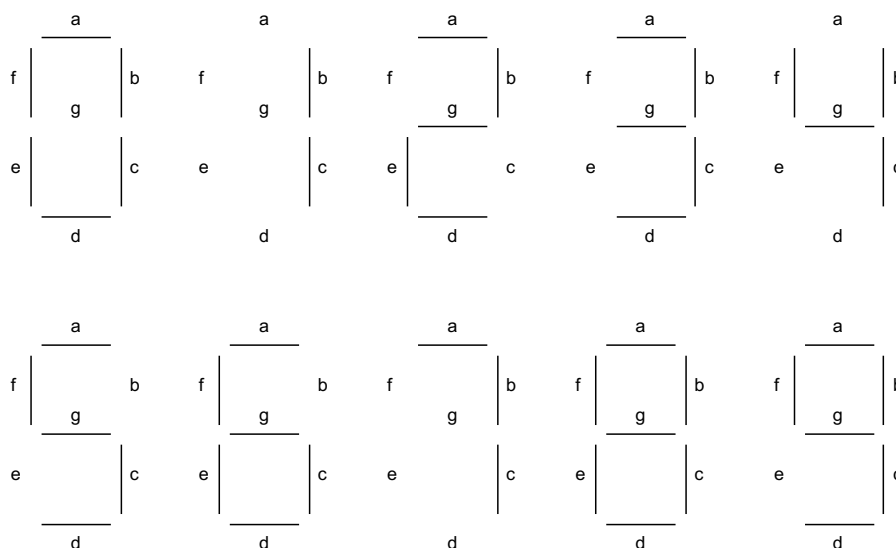
De plus, nous avons une entrée de sélection de polarité « Pol » :

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.

7-Segments Decoder - BCD (0..9) only



---

## 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)

- > 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.

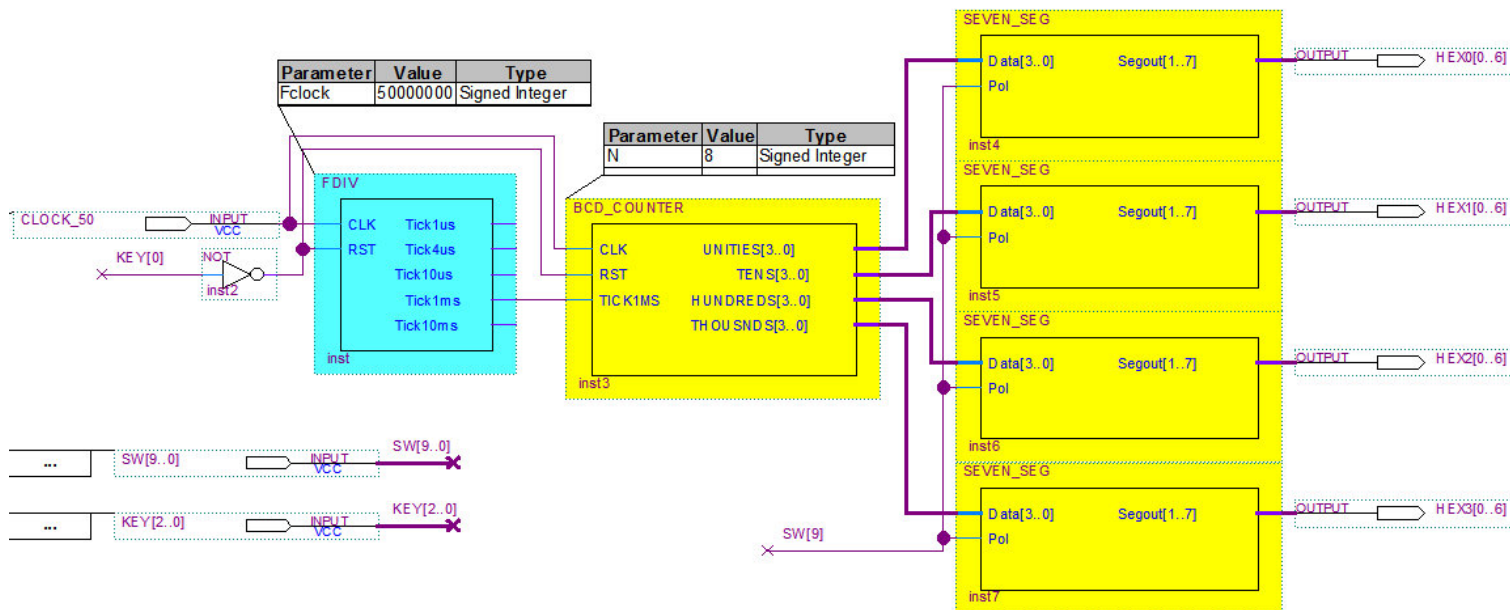
### Écrire le décodeur 7-segments

- > 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 **Assignments -> 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 Programmeur qui se trouve dans **Tools -> Programmer**.
- Connecter la carte de développement avec le câble 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 :**

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 :**

- Pour l'instant, la vitesse d'incréméntation 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'incréméntation du compteur à partir des switch SW0 à SW7. Ainsi, on pourra régler une vitesse d'incréméntation entre 1 et 256 ms. Il faudra incréménter 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)

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

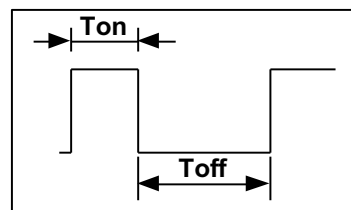
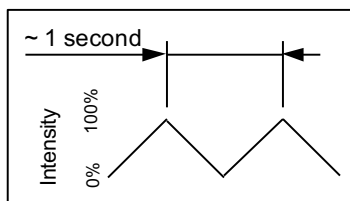
## 2/ PWM et LED qui bat (HeartBeat)

### 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 lumineuse 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-dessous.



L'intensité moyenne est donc proportionnelle au rapport cyclique  $\text{Ton} / (\text{Ton} + \text{Toff})$ . On peut donc faire varier **Ton** en gardant **Ton+Toff** constant. Un codage sur 8 bits est plus que suffisant (5 bits auraient suffi 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 ( $\text{Ton} + \text{Toff} < 20 \text{ ms}$ ) 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  $2.5 \text{ ms} / 256$  soit 10  $\mu\text{s}$ . 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  $1 / 512 = 2 \text{ ms}$ , 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  $\mu\text{s}$ , 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  $512 * 256 * 10 \mu\text{s} = \sim 1,33 \text{ seconde}$ , 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 Programmeur qui se trouve dans **Tools -> Programmer**.
- Connecter la carte de développement avec le câble 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 les 10 LEDs**

Pour cet exercice, vous devez réaliser une description qui permet d'afficher un chenillard sur les 10 LEDs de la carte FPGA.

Un chenillard consiste à allumer une seule LED à la fois et qui se déplace de gauche à droite avec une fréquence de 10 Hz. C'est-à-dire que chaque LED doit rester allumé pendant 100 ms.

L'application doit commencer par allumer la LED(0) et toute les autres LEDs éteintes. Après 100 ms, allumer la LED(1) et éteindre toute les autres LEDs. Après 100 ms allumer la LED(2) et ainsi de suite... LED(3), LED(4), etc...

Lorsque la LED(9) est allumé, il faut aller dans l'autre sens, après 100 ms allumer la LED(8) pendant 100 ms, LED(7), LED(6), etc...

Lorsque la LED(0) est allumé, il faut repartir dans l'autre sens et ainsi de suite.

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/ Jeu de Ping-Pong**

1. Le jeu de Ping Pong consiste à faire défiler les 10 leds comme un chenillard mais à chaque fois que la dernière LED (de droite ou de gauche) est allumée il faut que l'un des deux joueurs appui sur le bouton poussoir correspondant (KEY2 pour le joueur de gauche et KEY1 pour le joueur de droite) pour que la balle (le défilement des LEDs) reparte dans l'autre sens.

Dans un premier temps, vous n'avez pas à gérer le nombre de service par joueur. Faites-en sorte que celui qui a perdu, perd son service.

2. La principale modification à apporter consiste à faire défiler les LEDs de plus en plus rapidement au fur et à mesure de l'échange. La vitesse est réinitialisée à chaque changement de sens, c'est-à-dire à chaque fois qu'un joueur a perdu son service.



3. Afficher le score de la partie de Ping-Pong sur les afficheurs 7 segments. Vous avez le choix de la manière d'afficher les résultats de la partie.
4. Complétez votre code afin de compter les services pour que chaque joueur effectue 5 services et laisse ensuite l'autre joueur servir.