Informatique embarquée
TD — Interruptions

Objectifs
- Comprendre comment fonctionnent les interruptions
- Savoir utiliser les interruptions sur MBED

Matériel requis.
- Une plateforme nucleo-board STM32F446RE
- Une carte fille avec des LEDS

1 Commander une LED *via* une interruption.

On désire allumer la LED de la carte Nucleo-64, `LED1`, après avoir pressé une fois le bouton utilisateur `USER_BUTTON`. Cependant, on ne veut pas avoir à scruter en permanence l'état du bouton pour le faire.

1. Importer le projet MBED qui se trouve à l'adresse : `https://os.mbed.com/users/szuckerman/code/STM32_Button_Interrupt/`
   - Quelle est la source d'interruption ?
   - Que réalise le programme ? (indice : tout est dans le titre de cet exercice...). Spécifiquement : décrire pas à pas ce que fait le programme, ses fonctions, *etc.*
2. Modifier le programme pour qu'il compte le nombre de fois que le bouton `USER_BUTTON` a été pressé.
3. Créer un nouveau programme, et y copier le code permettant de réaliser un chenillard (voir TD GPIO). Le modifier pour que, à chaque pression du bouton `USER_BUTTON`, *et en utilisant les interruptions*, la fréquence de passage d'une LED à une autre s'accélère. Au bout de 16 pressions, on revient à la vitesse initiale de 1s.

2 Communications inter-cartes

Cet exercice est à faire avec deux cartes Nucleo-64. On désire piloter une carte Nucleo-64 à partir d'une autre Nucleo-64. Pour ce faire, on doit connecter les deux cartes à travers des broches, comme indiqué sur la Figure 1. L'objectif est le suivant :

1. Programmer la première carte pour qu'à chaque fois que le bouton utilisateur (broche `PC_13`) est pressé, on inverse l'état logique de la broche `PC_6`. Il faut que le bouton génère une interruption.
2. Programmer la deuxième carte pour qu'à chaque fois qu'un front montant est détecté sur la broche `PC_8`, on inverse l'état de la led (broche `PA_5`).

Question bonus : utilisation d'un minuteur/timer. On veut modifier le programme de la première carte de la manière suivante :

1. Déclarer une variable de type `Ticker` (on pourra l'appeler `g_timer` par exemple). On lui associera une fonction `void square_wave()`.
2. `void square_wave()` inversera l'état de la broche `PC_6` à chaque appel.
   — Pour faire ainsi, il faut écrire l'appel : `g_timer.attach(&square_wave, d);`
   — Ici, `d` représente le délai qu'on veut programmer dans le minuteur (par exemple, 0.5 seconde).
3.

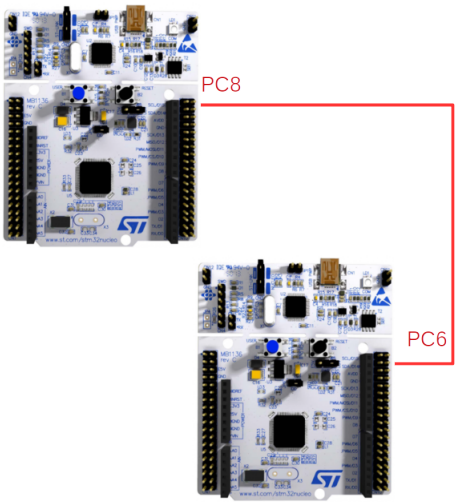

FIGURE 1 – Schéma de connexion entre 2 cartes STM32 pour l'exercice 1.

Annexe — Liste de fonctions utiles

— `void wait(float delay);` force le micro-processeur à attendre `delay` micro-secondes avant d'exécuter l'instruction suivante dans le programme.
— `fall(`*`nom_fonction`*`),rise(`*`nom_fonction`*`))` : fonctions rattachés à la variable qui « capture » les interruptions, et qui indiquent si on capture sur un front montant ou descendant.
   — Utilisation : `variable.rise(&nom_fonction);`
   — Utilisation : `variable.fall(&nom_fonction);`
— `attach(`*`nom_fonction`*`)` : fonction associée à certains types de variables, par exemple, `ticker` et `Serial`, et qui permettent de lever une interruption lorsqu'un événement survient, et d'appeler `nom_fonction`
   — Utilisation : `variable.attach(&nom_fonction);`

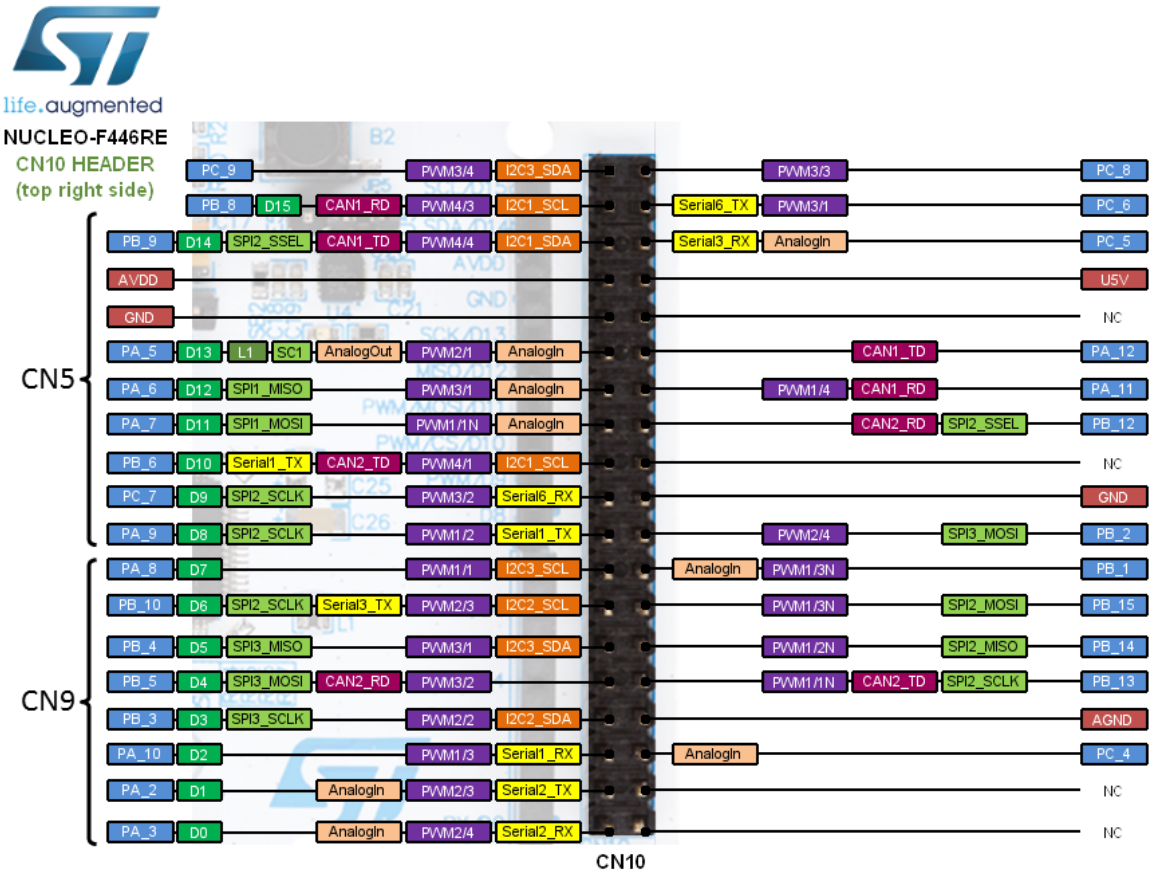

FIGURE 2 – Schéma des entrées-sorties pour la carte Nucleo-64.