TD 2 - ROVIN
Mesure de réflexe gérée en IT

L'objectif est ici de développer (compléter…) une application utilisant :
- le **port SPI**,
- différentes **interruptions** externes liées à des boutons,
- un **timer** avec l'interruption associée.

Cette application simule le fonctionnement d'une machine de mesure de réflexe. Le mode DEBUG de l'environnement sera utilisé pour finaliser l'application.

L'interface utilisateur de cette application est la suivante :
* LEDs rouges : valeur sur 8 bits binaire de la durée mesurée du réflexe (en ms). Ces LEDs sont contrôlées via le port SPI.
* LEDs jaunes : affichages de contrôle
  - LED7 : GameOver, quand il n'y a plus de parties jouables,
  - LED6 : Ready, retour de contrôle du bouton Ready,
  - LED[5:3] : GO !, pour dire au joueur que la mesure commence,
  - LED[2:0] : Nombre de parties restantes à jouer.
* Bouton 7 : Money, simule l'ajout d'argent (redonnant alors 1 partie par défaut)
* Bouton 6 : #Game, augmente le nombre de parties (par défaut 1 partie, extensible à 7)
* Bouton 5 : Ready, signale à la machine que l'utilisateur est prêt à jouer,
* Bouton 4 : Stop, arrête le compteur temporel de mesure de réflexe.

Le principe de jeu est simple : l'utilisateur insère de l'argent (bouton Money), puis signale qu'il est prêt à procéder à une mesure de réflexe (appui sur Ready). Après une durée aléatoire (entre 1 et 10 secondes), la machine affiche « GO ! » en démarrant simultanément un chronomètre logiciel. L'utilisateur appuie ensuite le plus rapidement possible sur le bouton Stop. La machine affiche alors son score. Un nouvel appui sur Ready provoque la relance du jeu (si le nombre de parties restantes est suffisant, sinon, la machine passe en GameOver).

Cette application de mesure de réflexe est programmée sous la forme **d'une machine d'états** dans laquelle les changements d'états se font principalement lors des **déroutements d'IT** (au sein de l'ISR IRQ_EVENT_).

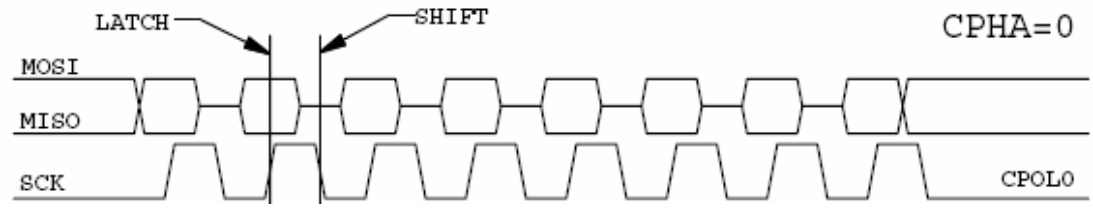
Les 5 états séquentiels de cette machine sont : Init, Attente_Money, Attente_Ready, Attente_Aléa et Attente_Stop. Après l'appui sur le bouton Stop, la machine d'états retourne en Attente_Ready s'il reste des parties, ou en Attente_Money sinon. En parallèle de cette exécution, les affichages sont mis à jour régulièrement.

PARTIE 1

Cette première partie vise à développer l'application avec SPI et interruptions externes (mais sans utiliser de timer). Le fichier TD2_etudiant.c du répertoire TD2, est une (bonne) base de travail…

1) Mise en place du transfert SPI pour utilisation des LEDs rouges

Contrairement aux LEDs jaunes de la carte, les LEDs rouges NE sont PAS connectées directement sur les broches externes du ROVIN. Ces dernières sont branchées sur un registre à décalage « Serial In Parallel Out » (74HC595) placé à côté du ROVIN et relié à ce dernier via une liaison série SPI.
Créer les deux fonctions suivantes et les tester dans un « petit » projet indépendant :
`void LED_R(unsigned char Value)` : Affichage de la valeur (1 octet) sur les LEDs
`void LED_R_Init(void)` : Création d'un port SPI (Cf. p160 dans l'aide du ROVIN), suivi de l'extinction des LEDs.
Les paramètres du port SPI sont les suivants : Port SPI n° 0, sur port PC du ROVIN, avec signal MOSI sur PC.3 initialisé à 1, signal MISO sur PC.2 (non initialisé car pas utilisé ici : les LEDs ne renvoient pas d'informations au maître), signal d'horloge SCK sur PC.1 initialisé à 1, signal de sélection d'esclave CS sur PC.0 initialisé à 0. De plus, le SPI doit fonctionner en BigEndian à la vitesse maximale, avec le mode de déclenchement (trigger) suivant : LATCH sur front montant d'horloge et avant le SHIFT :



Enfin, il faut savoir qu'une LED ne peut s'allumer que, premièrement, si la valeur 0 est mise sur le bit qui la contrôle et, deuxièmement, si le circuit 74HC595 a reçu une impulsion haute sur son signal de sélection d'esclave CS (Cf. p169).

2) Installation des boutons Ready et #Game

Compléter le code source du programme principal (dans le main) pour configurer correctement les IT sur les boutons Ready et #Game. Modifier également l'ISR concernant la section dédiée à la gestion du bouton #Game.
Pour cela, il faut remarquer que, par défaut, le nombre de parties jouables est 1. Ainsi, aucun appui sur le bouton #Game sera sanctionné par un GameOver immédiatement après l'unique mesure de réflexe.

Le bouton #Game est utile à l'utilisateur pour augmenter le nombre de parties. Ce dernier, après avoir appuyé sur Money, peut choisir de 1 à 7 parties. Il lui suffit de laisser le doigt appuyé sur ce bouton (auto-incrémentation toutes les 300ms en rotation : 1, 2,…, 6, 7, 1, 2, …). Dès le lancement de la première partie, l'utilisateur ne peut plus, bien sûr, changer le nombre de parties.

3) Traitement associé à ETAT_ATTENTE_ALEA

L'état d'attente d'une durée aléatoire est scindé en deux parties : la première partie est écrite dans l'ISR, et la seconde, dans le programme principal.
Compléter cette seconde partie, afin que la machine d'état puisse correctement passer dans l'état suivant (d'attente d'appui sur le bouton STOP).

A ce niveau de développement, l'application **doit fonctionner correctement**…

PARTIE 2

Dans cette seconde partie, l'application est modifiée afin d'installer un timer.

4) Installation d'un chien de garde

Le chien de garde sur la mesure de réflexe qu'il faut mettre en place est le suivant : si l'utilisateur attend trop longtemps (plus de 250ms) avant d'appuyer sur Stop, lui signifier qu'il a mis trop de temps à réagir…
Pour l'informer de ce débordement, éteindre partiellement les LEDs GO (LED_GO0 et LED_GO2, en laissant la LED_GO1 allumée) et modifier son score pour que la valeur présentée sur les LEDs soit 0.
Pour cette question, exploiter le timer T0.

5) Réalisation du clignotement des LEDs

Afin de rendre plus expressif le caractère « débordement du chien de garde » de la question précédente, il est demandé maintenant de faire clignoter les LEDs GO (plutôt que de les éteindre). Ce clignotement doit être réalisé avec une période de 200ms.
Pour parvenir à cela, utiliser le **même** timer T0 (qui sera ainsi bien rentabilisé).