

**Compte-rendu de TD1**

(Date : 13/02/24)

Etudiants : Malek SAHLI

Cursus : Master 1 LBD

**Année universitaire 2023/2024**

Exercice1 :

/\* Infinite loop \*/

  /\* USER CODE BEGIN WHILE \*/

  while (1)

  {

    /\* USER CODE END WHILE \*/

    /\* USER CODE BEGIN 3 \*/

    //HAL\_GPIO\_TogglePin(LD2\_GPIO\_Port, LD2\_Pin);

        //HAL\_Delay(500);

        if(ledOn){

          HAL\_GPIO\_WritePin(LD2\_GPIO\_Port, LD2\_Pin, GPIO\_PIN\_SET);

        }else{

          HAL\_GPIO\_WritePin(LD2\_GPIO\_Port, LD2\_Pin, GPIO\_PIN\_RESET);

        }

  }

  /\* USER CODE END 3 \*/

}

/\* USER CODE BEGIN 4 \*/

void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin){

  if(GPIO\_Pin == USER\_PB\_Pin){

    ledOn = !ledOn;

  }

}

/\* USER CODE END 4 \*/

La première section (/\* USER CODE END WHILE \*/ à /\* USER CODE BEGIN 3 \*/) est généralement réservée pour le code utilisateur exécuté à l'intérieur de la boucle while, entre chaque itération.

À l'intérieur de cette section, vous avez un bloc de commentaires vides. Cela indique que cette partie du code n'est pas utilisée dans cet exemple.

Ensuite, vous avez un bloc de code commenté qui utilise HAL\_GPIO\_TogglePin pour faire clignoter une LED à une fréquence de 500 ms. Cependant, ce code est actuellement commenté (//) et donc il n'est pas exécuté.

Après cela, il y a une section de code actif qui vérifie la valeur de la variable ledOn. En fonction de cette valeur, la LED connectée au port LD2\_Pin est allumée (GPIO\_PIN\_SET) ou éteinte (GPIO\_PIN\_RESET). Cela semble être un moyen de contrôler l'état de la LED en fonction de la valeur de ledOn.

Enfin, il y a une fonction HAL\_GPIO\_EXTI\_Callback qui est appelée lorsqu'une interruption est déclenchée sur la broche USER\_PB\_Pin. Dans cette fonction, la variable ledOn est inversée chaque fois que l'interruption est déclenchée sur cette broche. Cela suggère que le bouton utilisateur est utilisé pour basculer l'état de la LED entre allumé et éteint.

En résumé, ce code semble contrôler l'état d'une LED en fonction de la valeur de ledOn, qui peut être modifiée par une interruption déclenchée par le bouton utilisateur.

Exercice2 :

RCC->AHB2ENR |= RCC\_AHB2ENR\_GPIOAEN;

// Configure PA5 as output

GPIOA->MODER &= ~GPIO\_MODER\_MODE5\_Msk;

GPIOA->MODER |= (0x01 <<GPIO\_MODER\_MODE5\_Pos);

Les registres AHB2ENR, MODER et ODR sont des registres de contrôle utilisés dans le cadre du développement sur microcontrôleurs STM32, comme le STM32L4 dans votre cas. Chaque registre contrôle un aspect spécifique de la configuration ou du fonctionnement des périphériques intégrés au microcontrôleur, tels que les GPIO (General Purpose Input/Output).

AHB2ENR (AHB2 Peripheral Clock Enable Register):

Ce registre est utilisé pour activer ou désactiver l'horloge pour les différents périphériques présents sur le bus AHB2 (Advanced High-performance Bus 2). Dans votre code, vous activez l'horloge pour le périphérique GPIOA en configurant le bit correspondant dans ce registre. L'activation de l'horloge est nécessaire pour que le périphérique fonctionne correctement.

MODER (GPIO Mode Register):

Ce registre contrôle le mode de fonctionnement des broches GPIO. Chaque broche peut être configurée individuellement comme une entrée, une sortie ou dans différents modes alternatifs en utilisant ce registre. Dans votre code, vous configurez la broche PA5 (port A, broche 5) comme une sortie en effaçant les bits correspondants dans ce registre et en définissant les bits appropriés pour sélectionner le mode de sortie.

ODR (Output Data Register):

Ce registre est utilisé pour écrire des données aux broches de sortie GPIO. Chaque bit du registre correspond à une broche GPIO spécifique, et l'écriture d'un bit à 1 met la broche correspondante à l'état logique haut (Vcc) tandis que l'écriture d'un bit à 0 la met à l'état logique bas (GND). Dans votre code, vous utilisez ce registre pour faire clignoter la LED connectée à la broche PA5 en inversant l'état logique de la broche à chaque itération de la boucle principale.

Exercice3 :