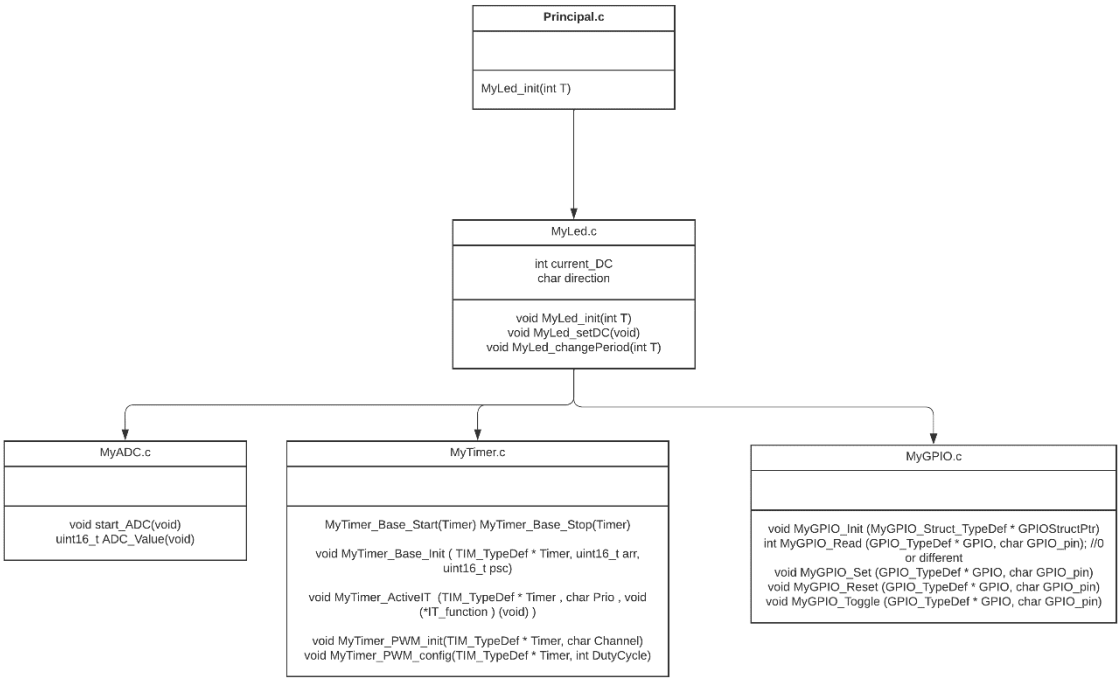
INSA TOULOUSE
2023 - 2024

PROJET MICROCONTROLEURS
PWM ET ADC

CRISTIAN MARTINEZ

LUZ VERA

1. DIAGRAMME DE CLASSES



2. FONTIONNEMENT DU CODE

Étape 1 : Configuration de la LED avec PWM

Le fichier principal "**Principal.c**" initialise la LED à l'aide de la fonction `**MyLed_init(int T)**`. Cette fonction est définie dans les fichiers d'en-tête "**MyLed.h**" et "**MyLed.c**".

La LED est connectée à la broche 0 du port GPIOA, tandis que le potentiomètre est connecté à la broche 1 du même port. La **PWM** est réalisée à l'aide du Timer2 (`TIM2`) sur le canal 0 (`**Channel_PWM**`).

La fonction `**MyLed_init(int T)**` configure la LED en calculant le temps d'interruption pour changer le cycle de travail du PWM 40 fois par période (T). Pour ce faire, divisez le temps de la période T par 40. Avec un ARR préalablement choisi et fixé, calculez la valeur du PSC nécessaire pour obtenir le temps d'interruption souhaité.

L'interruption du Timer3 (`TIM3`) est configurée pour changer le cycle de travail de la LED. La fonction `**MyLed_setDC()**` augmente ou diminue le cycle de travail de 5 % à chaque appel, générant une variation d'intensité lumineuse. Cette fonction est appelée 40 fois par période, comme indiqué ci-dessus.

La **fonction`MyLed_setDC()`** vérifie l'état actuel du cycle de travail **(current_DC)** et de la direction (direction). Selon ces valeurs, elle ajuste le cycle de travail de la LED de manière à obtenir une variation d'intensité lumineuse graduelle. Le cycle de travail est modifié par incréments de 5%, et la fonction veille à ce qu'il reste dans la plage valide (entre 0% et 100%). Enfin, la nouvelle valeur du cycle de travail est configurée dans le Timer PWM (Tim_PWM).

Étape 2 : Ajout d'un potentiomètre

La fonction `**MyLed_init(int T)**` est étendue pour prendre en compte un potentiomètre de 10k connecté à la broche 1 de GPIOA. L'ADC est utilisé pour mesurer la valeur du potentiomètre, et la période `T` est ajustée en conséquence. Pour ce faire, nous ajoutons la ligne **start_ADC()** et la boucle infinie où la valeur convertie par l'ADC est constamment lue avec la fonction **ADC_value()** et ensuite, la période est modifiée avec la fonction **MyLed_changePeriod(int T).**

La fonction **MyLed_changePeriod(int T),** recalcule la valeur de PSC nécessaire pour calculer le temps désiré de l'interruption qui modifie le cycle de travail de la led. Ensuite, avec la fonction **MyTimer_Base_Init()** on modifie ce temps avec le même ARR fixe et la nouvelle valeur PSC.

Étape 3 : Configuration du ADC, GPIO et le TIMER :

- **MyADC:**

La classe **MyADC** est responsable de la configuration et de la gestion de l'ADC (Analog-to-Digital Converter) sur les microcontrôleurs STM32. Elle fournit des méthodes pour initialiser l'ADC, activer l'horloge, configurer la fréquence d'horloge pour être inférieure à 16 MHz, et effectuer des conversions analogiques en numérique.

La méthode **start_ADC()** prépare l'ADC à lire des valeurs en activant le périphérique, en configurant une seule conversion, en ajustant le temps d'échantillonnage pour le canal choisi, et en démarrant la conversion. En option, elle peut activer une interruption à la fin de la conversion.

La méthode **ADC_Value()** démarre la conversion si elle n'est pas déjà en cours et renvoie la valeur lue, bloquant jusqu'à ce que la conversion soit terminée.

- **MyGPIO**:

La classe **MyGPIO** offre des fonctions pour initialiser et manipuler les broches des ports d'entrée/sortie général (GPIO) sur un STM32. Elle permet de configurer des broches individuelles avec des modes spécifiques (sortie, entrée, alternatif, etc.), de lire l'état des broches d'entrée et de changer l'état des broches de sortie.

Les méthodes incluent l'activation de l'horloge pour le port GPIO correspondant, la configuration des registres **CRL/CRH** pour définir le mode et la vitesse des broches, et l'initialisation des états de sortie avec des valeurs de résistance de **pull-up ou pull-down**.

Les fonctions **MyGPIO_Set**, **MyGPIO_Reset**, et **MyGPIO_Toggle** permettent de manipuler les états des broches de sortie.

- **MyTimer**:

La classe **MyTimer** gère les timers intégrés pour le microcontrôleur STM32, offrant des fonctionnalités pour l'initialisation basique des timers, la configuration pour la modulation de largeur d'impulsion (**PWM**), et la gestion des interruptions. Elle permet de définir des périodes pour les timers et d'activer des interruptions avec une priorité définie pour les événements de timer comme l'expiration du compteur.

Les fonctions **MyTimer_Base_Init** et **MyTimer_ActiveIT** sont utilisées pour initialiser les timers avec des valeurs spécifiques pour **ARR** (Auto-Reload Register) et **PSC** (Prescaler), et pour activer les interruptions.

En plus, **MyTimer_PWM_init** et **MyTimer_PWM_config** sont dédiées à la configuration du PWM pour contrôler par exemple la luminosité d'une LED.