Activité 2 : Le GPIO et les librairies LL (STMicro)

Periph'Team - INSA de Toulouse

1 Contexte matériel

Le TP s'inscrit bien sûr dans le cadre du chronomètre. L'*ibd* (schéma montrant la structure matérielle du chronomètre) que l'on peut voir sur le document de présentation (page 4) montre bien le rôle des *GPIO* dans le projet. On ne va s'intéresser qu'aux *GPIOs* qui gèrent les boutons poussoirs et la LED. Les 3 pins sont réunies sur le port *GPIOC*.

1.1 La carte Nucléo avec repérage des IO utiles

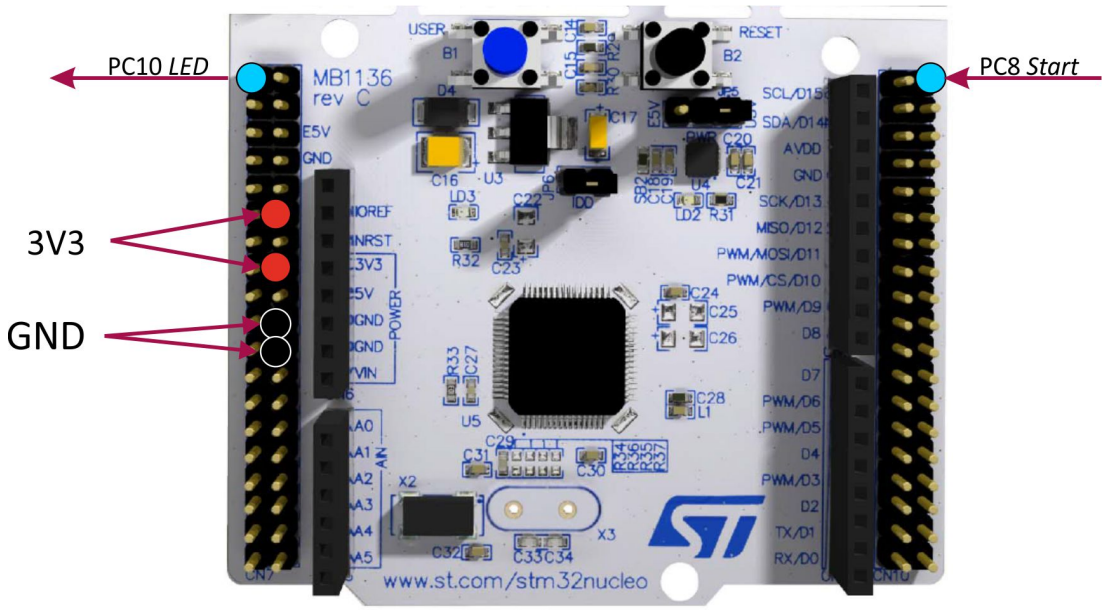

Figure 1: *Les broches utilisées*

1.2 Contraintes IO

Contraintes sur les entrées

- BP Stop/Start : *PC8* en **floating input**
- BP Reset-Chrono : *user button* (bleu) de la *Nucleo*

Contraintes sur les sorties

- LED PC10 en **output pushpull**
- LED PC10 en **output open drain** (dans un second temps)

1.3 Le matériel électronique

- LED
- BP (Bouton poussoir),
- résistance 10kΩ,
- fils noir, rouge, jaune (avec résistance interne de 100Ω pour prévenir des court-circuits),
- mini plaque d'essais.

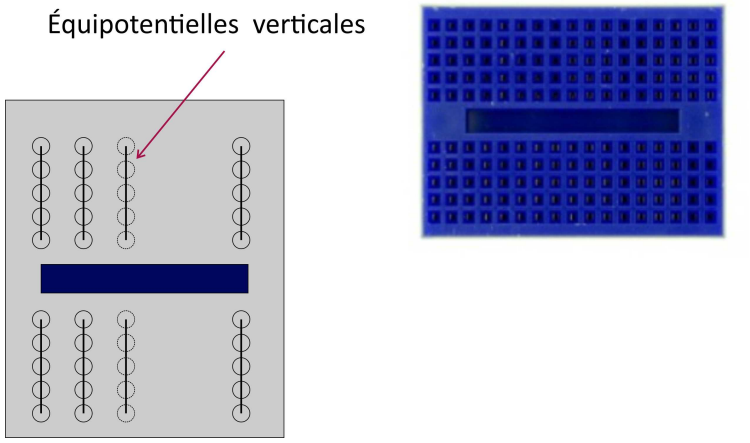

Figure 2: *La plaque d'essais miniature*

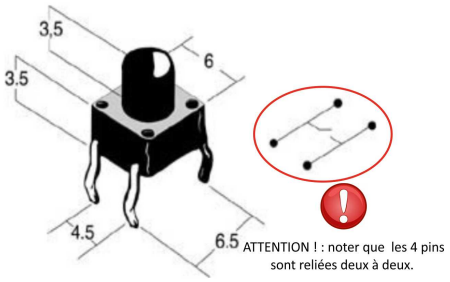

Figure 3: *Le bouton poussoir à implanter sur la plaque d'essais .*

2 Travail à faire

- Lire les documents, vidéos ... concernant les *GPIO*. Dans un premier temps, on s'intéressera à l'aspect électrique des ports et non à la programmation. A l'issue de vos lectures, cours, visionnages... vous devez avoir compris :

- ce qu'est globalement un *GPIO* et à quoi il sert,
- la différence entre une entrée *floating input*, *pull-up* ou *pull-down*, ce que cela implique en terme de circuit d'entrée extérieur au µC,
- la différence entre une sortie *push-pull* et une sortie *open drain* , que cela implique en terme de circuit de sortie extérieur au µC,
- la subtilité du mode *analog input*, son avantage par rapport au floating input,
- les configurations en *alternate function*.

2.1 Schéma électronique

La datasheet du *STM103* nous dit que :

- en mode *floating input*, le courant d'entrée $\pm 1 \mu A$,
- en mode *pull-up / pull-down* les résistances sont de $40 k\Omega$ (valeur typique),
- en mode *output push-pull* le courant de sortie évolue dans la fourchette $\pm 8 mA$ (maximum par sortie, si plusieurs sorties sont utilisées il faut aussi tenir compte d'une limite de $150 mA$ sur le courant total sur la masse (Vss) et l'alim (Vdd)),

Un niveau de tension supérieur à $1,5V$ environ sur la pin du *GPIO* est interprété par un niveau logique 1. Un niveau de tension inférieur à $1,1V$ environ sur la pin du *GPIO* est interprété par un niveau logique 0.

► Tracez sur papier le schéma électronique des 3 circuits d'entrée et du circuit de sortie, à partir des éléments dont vous disposez. Pour le bouton *reset*, voir le schéma électronique de la carte *Nucléo* disponible sous Moodle. Discutez avec vos enseignants de vos solutions.

2.2 Programmation niveau registre

► Reprendre votre projet (faites en une copie et renommez le *BacASable*...). En effet dans cette partie vous allez expérimenter juste le bouton *Start* et la *LED*. La programmation se fera salement dans le main ! Faire un code minimaliste qui :
  - configure *Start* en *floating input*,
  - configure la *LED* en *output push pull*,
  - fasse en sorte que lorsque *Start* est appuyé la *LED* s'allume.

► Essayez votre code en simulation d'abord puis en réel avec le matériel dont vous disposez,
► Modifiez votre code : on souhaite maintenant avoir une configuration open drain,
► Procéder à la vérification pratique.

2.3 Programmation en utilisant la libraire *LL*, reprise du projet

Jusqu'ici, vous avez utilisé directement les registres pour obtenir le fonctionnement souhaité d'un périphérique. À partir de maintenant nous allons utiliser une série de bibliothèques, dite librairie *LL (Low Layer)*. Ces bibliothèques sont fournies sous forme de code source C par le constructeur *ST*, dans la section Drivers du *package STM32CubeF1* (*F1* pour la famille *STM32F1xx*). Vous n'aurez plus besoin de faire directement des opérations de masques, mais les concepts demeurent. Vous allez naviguer dans des *.h* un peu chargés, mais avec un peu de méthode vous arriverez vite à les prendre en main. Sur le site *Moodle*, on trouvera des documents qui aident à découvrir / utiliser les librairies *LL*.

2.3.1 Nouvelle structuration du projet

Voici la nouvelle structure de projet proposée

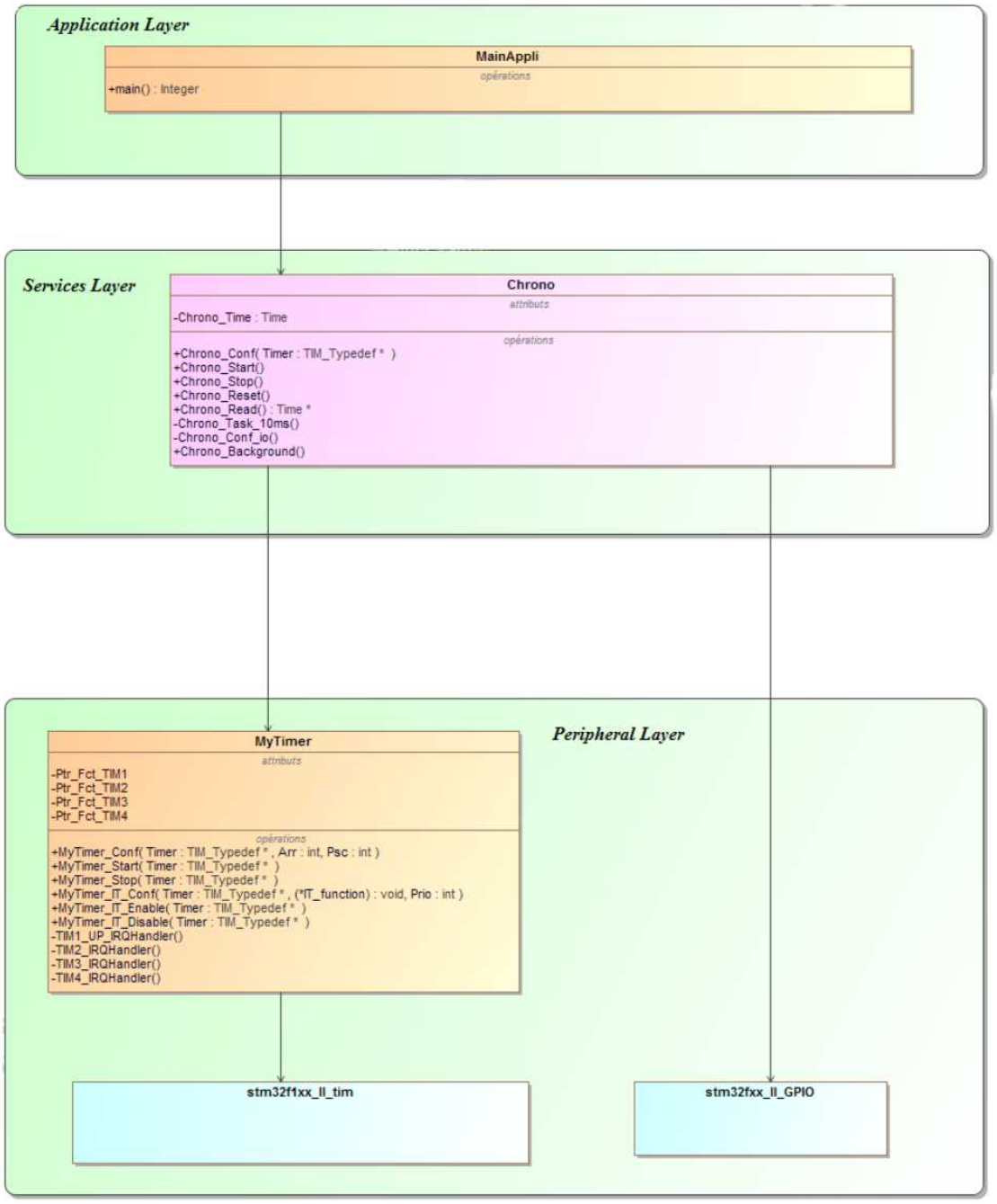

Figure 4: *Diagramme de classes du chronomètre, version LL, intégrant les GPIO*

2.3.2 Explication de la structuration :

Il s'agit d'une manière de faire, qui n'est pas forcément LA référence mais qui possède sa cohérence.

La couche périphérique Elle contient un ou deux fichiers selon la complexité du périphérique. Typiquement, un périphérique qui fait intervenir une interruption ou qui est relativement complexe justifie une couche périphérique supplémentaire, sorte d'API « more friendly » pour les couches service. Typiquement le timer est assez complexe (il y a beaucoup de modes non encore vus comme la PWM). Les périphériques de communication, *USART, SPI, I2C* demandent classiquement des fonctions de sortie ou d'entrée sous format string (genre printf, scanf...). Les librairies *LL* ne l'implémentent pas. Une librairie intermédiaire est alors la bienvenue. Par contre, les *GPIO* sont assez simples d'emploi, ainsi la communication entre le service Chrono et la librairie *LL* est directe.

La couche service Elle n'a pas beaucoup évolué. On voit deux fonctions en plus :

- *Chrono_Conf_io( )* : elle doit permettre de configurer les 3 IO pour recevoir les 2 BP et la LED. Elle est privée et doit être appelée par *Chrono_Conf( )*.
- *Chrono_Background( )* : celle ci est à appeler en permanence dans le main, à l'intérieur du *while(1)*. C'est la tâche de fond non bloquante du module chronomètre. Elle est la plus courte possible. Elle consiste juste à lire l'état des divers boutons puis de faire les actions associées (*start, stop, reset*).

La LED sera allumée puis éteinte dans la *callback Chrono_Task_10ms( )*, à une fréquence de 1Hz.

2.3.3 Préparation du projet

- Téléchargez le projet *ProjKEIL_Chrono_TIM_IT_LL*. Il s'agit en fait de votre projet entièrement revu au niveau du corps de *MyTimer.c*. Le *header* n'a pas été modifié pour conserver la compatibilité avec le module supérieur,
- Compilez faites l'édition de lien. Testez-le.
- Analysez le code modifié pour vous familiariser avec les librairies *LL*.

2.3.4 Codage

- Apportez les modifications nécessaires dans KEIL : ajoutez le module *LL GPIO* (il se trouve dans le répertoire *../LLDrivers/src*.
- Rédigez les fonctions manquantes, et modifiez *Chrono_Task_10ms( )* pour gérer la *LED*,
- Procédez au test en simulation : c'est possible avec l'interface périphérique *GPIO*,

2.3.5 Test matériel

- Câblez le bouton poussoir à la carte *Nucléo*, ainsi que la *LED*,
- Tester le chronomètre