Activité 1 : Notion de General Purpose Input Output - Ecriture d'un premier *driver*

Periph'Team - INSA de Toulouse

1 Objectifs pédagogiques

- ♢ Savoir programmer les**G**eneral **P**urpose **I**nput **O**utput (GPIO) au niveau des registres du processeur
- ♢ Savoir choisir un mode de configuration pour une application donnée
- ♢ Savoir construire une bibliothèque simple au niveau driver (driver)
- ♢ Savoir mettre au point un programme sous KEIL avec des outils de debug avancé

2 Contexte matériel

Ce TP va s'intéresser qu'aux *GPIOs*, c'est à dire en premier lieu à la capacité qu'a un μcontrôleur de gérer des signaux d'entrée/sortie binaire. Dans ce TP nous utiliseront simplement des boutons poussoirs et des LED pour mettre en œuvre ces principes.

2.1 La carte Nucléo avec repérage des IO utiles

2.2 Contraintes IO sur la carte Nucléo

Contraintes sur les entrées

- BP externe : à connecter sur *PC8*
- BP *user button* (bleu) de la carte *Nucleo*

Contraintes sur les sorties

- LED externe a connecter sur PC10
- LED verte de la carte de la *Nucleo*

2.3 Le matériel électronique fourni

- LED
- BP (Bouton poussoir) (Figure 3)
- résistance 10kΩ,

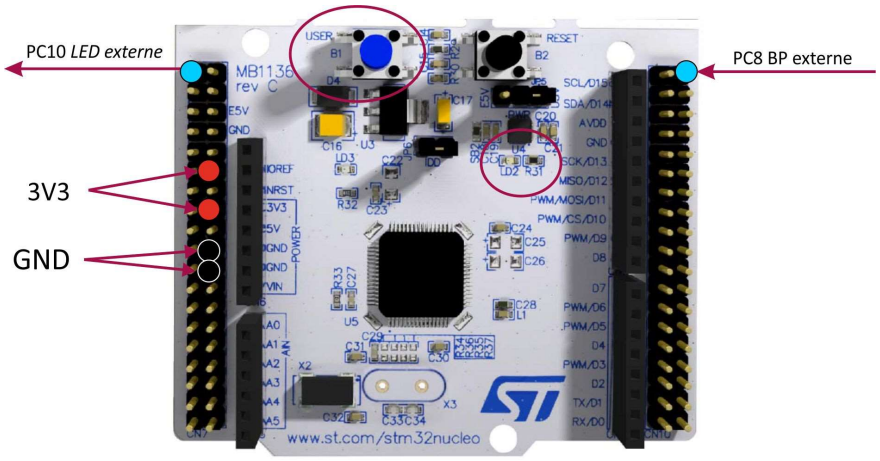

Figure 1: *Les broches utilisées*

- fils noir, rouge, jaune (avec résistance interne de 100Ω pour prévenir des court-circuits),
- mini plaque d'essais (Figure2).
- une carte de développement (Figure 1)

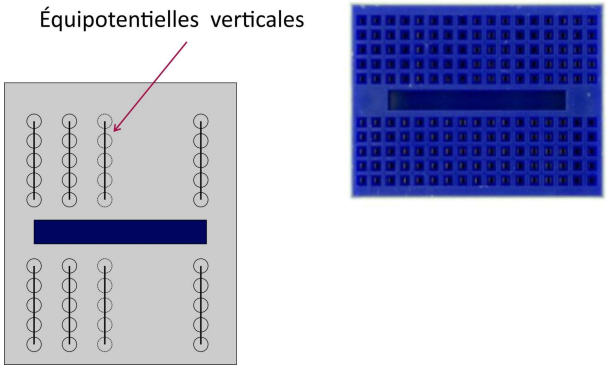

Figure 2: *La plaque d'essais miniature*

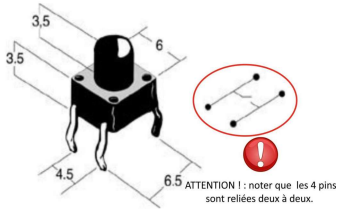

Figure 3: *Le bouton poussoir à implanter sur la plaque d'essais .*

2.4 Lien et contexte pour le projet voilier

Toutes les connexions avec la processus à commander, dans notre cas le voilier, passent par les GPIO. C'est évident et direct pour le cas d'une entrée/sortie binaire (un led, un poussoir) mais c'est également le cas pour utiliser les périphériques plus complexes (ADC, I^2C, PWM,...). Dans le projet que vous allez développer, la figure 4 montre

pour le voilier quelques-uns des besoins en terme d'unités périphériques et comment celles-ci communiquent avec l'extérieur via les GPIO. Cette liste n'est pas exhaustive.

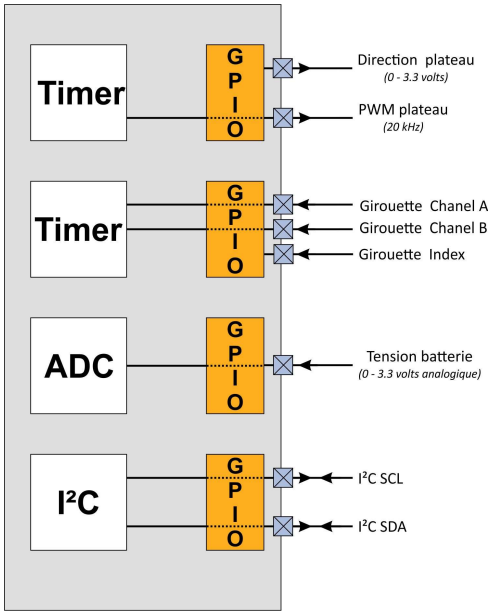

Figure 4: *Synoptique Périphérique et GPIO pour le voilier*

3 Travail à faire

3.1 Première version de l'application

Même si cette application n'aura pas d'utilité directe dans le projet final du voilier, elle va vous permettre d'appréhender le fonctionnement binaire d'une broche d'un GPIO.

On vous demande de réaliser l'allumage et l'extinction d'une led simple (Output) par recopie de l'état binaire d'un bouton poussoir (Input). On pourra donc au final visualiser sur la fenêtre *Logic Analyser* de l'IDE Keil le chronogramme de la figure 5.

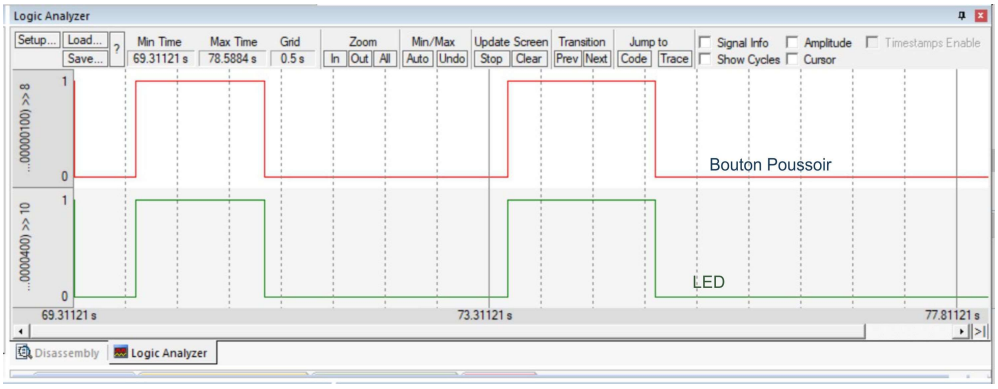

Figure 5: *Capture de la fenêtre Logic Analyser*

- Créez un premier projet "from scratch" en suivant la méthode exposée dans les vidéos suivantes :
  - *ScreenCast : Configuration d'un projet Keil "from scratch"*

  - *ScreenCast : Configuration d'une cible réelle*

- 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*.
- Jouer avec le "clicodrome" de la partie debug de l'IDE. Pour cela voici les étapes à suivre :
  - Ecrivez une application ne contenant que le code c minimal suivant :

```
#include "stm32f10x.h"
int main(void)
{
 RCC->APB2ENR |= (0x01 << 2) | (0x01 << 3) | (0x01 << 4);
        do
        {
        }while(1);
}
```

> *La seule ligne de code qui apparait dans ce listing à de quoi effrayer. Cependant ceci est loin d'être si compliqué. APP2ENR est un registre de l'unité périphérique* RCC *qui gère les horloges du µ contrôleur. La syntaxe C* RCC->APB2ENR *permet d'y accéder. On pourrait la remplacer (mais le code deviendrait encore plus obscur) par une affectation direct à l'adresse exacte d'implémentation de ce registre, à savoir :*
> $*(0 \times 40021018) | = (0 \times 01 << 2) | (0 \times 01 << 3) | (0 \times 01 << 4)$.
> *La partie droite de l'affectation utilise la technique des masques. Il s'agit d'une approche logicielle de modification de bits ou de champs de bits d'une valeur que nous vous demanderons d'utiliser quasi systématiquement par la suite.*

  - Quelle sera la valeur du registre (16 bits) APB2ENR après l'exécution de cette instruction ?
  - Créez l'exécutable **pour la cible réelle** de cette application et passez en mode *debug*.
  - Repérez sur la documentation propre à la *carte nucléo 103RB* , sur quels GPIO sont connectés la LED verte (Led2) et le bouton poussoir bleu (User Button)?
  - Ouvrez dans le menu *Peripheral->General Purpose IO*, le ou les ports concernés.
  - Modifiez les paramètres et les registres dans cette interface pour pouvoir lire l'état du poussoir bleu et allumer la Led verte.
  - Testez quand agissant sur les registres du GPIO sont bien en interaction avec les élements physiques de la carte.
  - Faites la même chose **en mode simulé** en vous posant la question de l'existence des pseudo-registres PINS qui apparaissent maintenant dans les fenêtres *Peripheral->General Purpose IO*.

3.2 Seconde version de l'application : ajout de composants

On vous demande maintenant d'utiliser une led et un bouton poussoir extérieurs à la carte. Ce matériel est fourni dans le kit. Cette partie pose la problématique du branchement électrique de ces composants passifs.

- Tracez sur papier le schéma électronique du circuit d'entrée et du circuit de sortie, à partir des éléments dont vous disposez. Discutez avec vos enseignants de vos solutions.
- Faites la même manipulation que la précédente en mode réel et vérifiez que vous êtes capable d'interagir avec ces deux composantes d'E/S.

3.3 Troisième version de l'application : ajout de code

Logiciellement pour l'instant le code ne fait quasiment rien (à part une initialisation visible et beaucoup d'autres cachées dans les fichiers startup). On vous demande donc d'écrire les quelques lignes de programme qui relient le bouton poussoir à la led. Dans cette version devra donc simplement recopier l'état du poussoir sur la led. En plus des registres *IDR* et *ODR* sur lesquels vous allez agir, interrogez-vous sur le rôle et l'utilité des registres *BSR* et *BSRR*.

- Faire un code minimaliste qui :
  - configure le *poussoir* en *floating input*;
  - configure la *LED* en *output push pull*;
  - fasse en sorte que lorsque le *bouton* 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.
- Procédez à la vérification expérimentale.

3.4 Programmation en créant un driver minimal

3.4.1 Principe et utilité

Jusqu'ici, vous avez utilisé directement les registres pour obtenir le fonctionnement souhaité d'un périphérique. À partir de maintenant vous allez écrire et utiliser une série de drivers. L'idée de base est que **seule la couche driver connait les registres**. Les niveaux supérieurs devront être d'un niveau d'abstraction tel qu'aucun accès direct au registre ne soit possible. Cette structuration permet idéalement de changer de μcontrôleur dans un projet en ne changeant que la couche basse, donc les drivers. Des bibliothèques de ce type existent sous forme de code source C et sont disponibles sur le site du constructeur *ST*. Cependant la finalité de ce module d'enseignement est que vous compreniez au mieux le fonctionnement des différentes unités périphérique et que vous soyez en mesure de développer ce genre de bibliothèques. Nous avons donc choisi de ne pas vous former à l'utilisation de ces outils.

3.4.2 Organisation de vos projets

Vos différents drivers vont évoluer et se compléter au fur et à mesure de la série de TP. Or ces drivers doivent être communs aux différents projets. Nous vous invitons à suivre l'exemple de le figure 6 pour organiser vos fichiers sur votre espace de stockage.

Par la suite, lorsque vous aurez un nouveau projet à créer, soit vous repartez sur un projet "from scratch" soit vous faites un copier/coller du répertoire contenant un projet et vous adaptez les fichiers sources et include du duplicata.

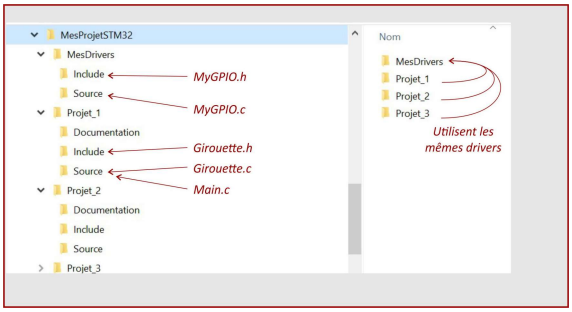

Figure 6: *Organisation des projets sur le disque*

3.4.3 Mise en oeuvre

Les fichiers *.h* sont essentiels pour une écriture rigoureuse en langage C. L'écriture de ces fichiers peut composer une étape du développement et peuvent aider à la spécification de ce qui est attendu. Dans le cas de ce premier driver nous nous demandons d'écrire le fichier *.c* correspondant à fichier *Driver_GPIO.h* suivant :

```
#ifndef MYGPIO_H
#define MYGPIO_H
#include "stm32f10x.h"

 typedef struct
{
        GPIO_TypeDef * GPIO;
        char GPIO_Pin;       //numero de 0 a 15
        char GPIO_Conf;     // voir ci dessous
} MyGPIO_Struct_TypeDef;

#define In_Floating      0x...  // a completer
#define In_PullDown     0x...       // a completer
#define In_PullUp       0x...   // a completer
#define In_Analog       0x...       // a completer
#define Out_Ppull       0x...       // a completer
#define Out_OD          0x...   // a completer
#define AltOut_Ppull    0x...   // a completer
#define AltOut_OD       0x...           // a completer

void MyGPIO_Init( MyGPIO_Struct_TypeDef *  GPIOStructPtr);
int MyGPIO_Read(GPIO_TypeDef * GPIO, char GPIO_Pin); // renvoie 0 ou autre chose different de 0
void MyGPIO_Set(GPIO_TypeDef * GPIO, char GPIO_Pin);
void MyGPIO_Reset(GPIO_TypeDef * GPIO, char GPIO_Pin);
void MyGPIO_Toggle(GPIO_TypeDef * GPIO, char GPIO_Pin);

#endif
```

- Créez un **nouveau projet distinct du projet précédent**, qui inclura quand il sera écrit ce driver.
- Ecrivez le corps des fonctions du driver dans le fichier *Driver_GPIO.c* (positionné au "bon endroit" dans vos répertoires)
- La ligne $RCC->APP2ENR| = (0 \times 01 << 2)|(0 \times 01 << 3)|(0 \times 01 << 4)$ fait référence à un registre. Il faut donc la déplacer dans le driver. Comprenez le rôle de ces bits d'enable et adaptez alors le code de votre driver.
- Modifiez le code du programme principal précédent pour appeler maintenant les fonctions de votre driver
- Faites un premier test pour constater le bon fonctionnement de cette version.

3.4.4 Test du driver

Pour être certains que votre driver fonctionne correctement il faudrait mettre en œuvre une gestion systématisée de test. Par manque de temps nous ne ferons pas cette campagne de test. Nous allons cependant essayer d'adapter cette version de l'application pour s'assurer qu'une grande partie des cas de fonctionnement soit couverte par des tests "manuel".

- Prévoir les lignes de code qui permettent :
  - de tester le driver pour chacun des ports A,B,C,D
  - pour au moins un des ports, tester au moins 3 pins (un en pf, un en PF et un au milieu)
  - pour ces pins, couvrir l'ensemble de configuration possible (input, output, analog,...)
- Lancez ce scénario de test et vérifiez "de visu" dans les interfaces du débogeur que l'action de configuration demandée se déroule correctement.