Laboratoire de microprocesseurs - n°1
GPIO, Alternate functions, modularisation du code

I Présentation du sujet

L'objectif de ce laboratoire est d'abord de développer une API (fichiers **io.h/.c**) afin de gérer les ports GPIO du microcontrôleur STM32F411 pour pouvoir facilement y lire ou y écrire une valeur. Dans un deuxième temps, on utilisera les fonctions alternatives (*alternate functions*) des GPIO pour transmettre un signal PWM afin de piloter la luminosité de leds. La préparation de ce laboratoire a été faite dans le TD1.

Dans ce laboratoire, on utilisera la led tricolore (led RGB) et le joystick, représenté par 5 interrupteurs (*switches*), de la carte d'extension Appshield. Une API sera également développée pour gérer ces ressources. Se reporter au TD pour retrouver les GPIO (port et broches) connectés aux leds et aux interrupteurs.

Une API permettant d'utiliser les timers TIM2 à TIM5 est fournie précompilée. Consulter son fichier d'en-tête **timer.h** pour prendre connaissance des fonctions de l'API. Une API de l'afficheur LCD est également fournie précompilée. Consulter son fichier d'en-tête **lcd.h**.

Les documentations relatives à l'utilisation du microcontrôleur STM32F411 sont disponibles sur les postes de travail et en ligne (*Moodle*).

II Modularisation du code

II.1 API io.h/.c (lib)

Comme cela a été vu en TD, la structuration du code permettra d'ajouter ou de retirer des coupleurs d'un projet sans remettre en cause le reste du code développé. Dans cette optique, on souhaite développer une API pour gérer les GPIO du microcontrôleur.

La fonction *io_configure*, dans le fichier **io.c**, est fournie complète. Les fonctions dont le prototype est donné ci-dessous sont à coder :

a) `uint32_t io_read(GPIO_t *gpio, uint16_t mask);`
lecture de la valeur sur le port `gpio` après masquage. Les bits correspondants à un 0 du masque sont lus comme des 0.

b) `void io_write(GPIO_t *gpio, uint16_t val, uint16_t mask);`
écriture de `val` sur le port `gpio` après masquage. Un bit à 1 (resp. 0) dans `val` écrit un niveau logique haut (resp. bas) sur la broche correspondante du port. Les bits à 0 du masque `mask` permettent de ne pas modifier la valeur des broches correspondantes. Aucun décalage n'est appliqué à `val` par la fonction.

c) `void io_write_n(GPIO_t *gpio, uint16_t val, uint16_t mask);`
écriture de `val` sur le port `gpio` après masquage. Un bit à 1 (resp. 0) dans `val` écrit un

niveau logique bas (resp. haut) sur la broche correspondante du port. Les bits à 0 du masque `mask` permettent de ne pas modifier la valeur des broches correspondantes. Aucun décalage n'est appliqué à `val` par la fonction.

d) **`void`** *`io_set`*`(`**`GPIO_t *`**`gpio, `**`uint16_t`** *`mask`*`);`
écrit un '1' sur les bits du port `gpio` identifiés dans le masque `mask`.

e) **`void`** *`io_clear`*`(`**`GPIO_t *`**`gpio, `**`uint16_t`** *`mask`*`);`
écrit un '0' sur les bits du port `gpio` identifiés dans le masque `mask`.

II.2 API leds.h/.c (libshield)

La led RGB faisant partie de la carte d'extension Appshield, son API **leds.h/.c** est placée dans le dossier libshield du projet. Les fonctions dont le prototype est donné ci-dessous sont à coder :

a) **`uint32_t`** *`leds_init`*`(`**`void`**`);`
configure les 3 broches GPIO connectées aux leds de la led RGB.

b) **`void`** *`leds`*`(`**`uint16_t`** *`val`*`);`
affiche sur les leds rouge, verte, bleue la valeur binaire passée en paramètre.

c) **`void`** *`red_led`*`(`**`uint32_t`** *`on`*`);`
allume la led rouge si le paramètre `on` n'est pas nul, l'éteint sinon.

d) **`void`** *`green_led`*`(`**`uint32_t`** *`on`*`);`
allume la led verte si le paramètre `on` n'est pas nul, l'éteint sinon.

e) **`void`** *`blue_led`*`(`**`uint32_t`** *`on`*`);`
allume la led bleue si le paramètre `on` n'est pas nul, l'éteint sinon.

II.3 API sw.h/.c (libshield)

Le joystick faisant partie de la carte d'extension Appshield, son API **sw.h/.c** est placée dans le dossier libshield du projet. Le prototype des fonctions de l'API à coder est donné ci-dessous :

a) **`uint32_t`** *`sw_init`*`(`**`void`**`);`
configure les 5 broches GPIO connectées aux interrupteurs du joystick.

b) **`uint32_t`** *`sw_input`*`(`**`void`**`);`
renvoie l'état des 5 boutons poussoirs, dans l'ordre, SW_CENTER, SW_DOWN, SW_UP, SW_LEFT, SW_RIGHT. La valeur retournée est alignée sur le LSB (SW_RIGHT). Un bit à '1' indique que le bouton poussoir est appuyé.

Rem. : l'API contient déjà les fonctions sw_right, sw_left, sw_up, sw_down et sw_center qui filtrent les rebonds des switches du joystick. Ces fonctions répondent '1' la première fois que l'interrupteur est reconnu appuyé stable.

II.4 Tests

Vérifier que le paramètre MAIN1 est défini (sélection du main() compilé) puis compiler et télécharger le projet.

Vérifier que le programme de test permet d'allumer successivement les leds rouge, verte et bleue en utilisant d'abord la fonction leds, puis les fonctions red_led, green_led et blue_led. Après le test des leds, l'activation d'un switch est indiqué sur l'afficheur LCD de la carte d'extension Appshield.

III Modulation PWM : pilotage d'une led

La mise en œuvre d'un modulateur PWM est réalisée par le programme MAIN2.

Le programme a pour but de piloter la led rouge par un signal PWM. Lorsque le joystick est actionné à droite, le rapport cyclique est diminué d'un pas de 10 %, lorsqu'il est actionné à gauche, le rapport cyclique est augmenté d'un pas de 10 %.

III.1 Initialisations

Le joystick peut être utilisé après appel de la fonction sw_init développée dans la partie précédente. Les leds nécessitent une initialisation différente de celle réalisée par la fonction led_init car une fonction alternative des GPIO est à utiliser pour transmettre le signal PWM. Les fonctions de l'API des timers sont utilisées pour la configuration et le pilotage du modulateur pwm.

a) retrouver le timer et le canal pwm pouvant piloter la led rouge. Compléter l'appel de la fonction io_configure pour que le signal pwm soit associé à la broche à laquelle la led rouge est connectée.
b) compléter l'appel des fonctions de l'API timer permettant :
   1. d'initialiser le pwm choisi avec une période "pwm_period" ms,
   2. d'autoriser le canal pwm choisi à générer un signal avec un rapport cyclique "dutycycle",
   3. de démarrer le timer choisi.

III.2 Pilotage de la led rouge

La fonction timer_wait_us est fournie de manière précompilée et permet de faire une pause dans le programme pendant la durée spécifiée (en µs). On l'utilisera pour l'appel périodique, toutes les "sampling_period" µs, des fonctions anti-rebonds sw_right et sw_left.

La fonction pwm_channel_set de l'API timer permet de changer la valeur du rapport cyclique d'un modulateur pwm. On l'utilise ici pour changer la luminosité de la led rouge.

a) compléter le code principal pour que la boucle infinie soit exécutée toutes les "sampling_period" µs,
b) détecter l'appui sur sw_right et sur sw_left :
   1. si l'appui est à droite, augmenter le rapport cyclique d'un pas de 10 %. Afficher la nouvelle valeur de "dutycycle" sur l'afficheur LCD,
   2. si l'appui est à gauche, diminuer le rapport cyclique d'un pas de 10 %. Afficher la nouvelle valeur de "dutycycle" sur l'afficheur LCD.
c) Tester le programme (vérifier que le paramètre MAIN2 est défini).

IV Modulation PWM : programme complet

La mise en œuvre du programme complet est réalisée par le programme MAIN3.

Le programme a pour but de piloter la led rouge et la led verte par des signaux PWM de rapport cyclique complémentaire. Par exemple, lorsque la led rouge est pilotée à 20 %, la led verte l'est à 80 %. D'autre part, en plus des interrupteurs droite/gauche, on souhaite ajouter des fonctionnalités supplémentaires : lorsque l'interrupteur haut est maintenu appuyé pendant plus d'une durée "sw_up_delay_base", le rapport cyclique "dutycycle" est augmenté d'un pas de 10 % périodiquement au bout de la même durée, tant que l'appui sur le bouton est maintenu. Le fonctionnement sera le même pour le bouton bas, avec une diminution du rapport cyclique d'un pas de 10 %.

a) retrouver le timer et le canal pwm pouvant piloter la led verte. Compléter le code pour que le deuxième signal pwm soit associé à la broche à laquelle la led verte est connectée.
b) compléter l'appel des fonctions de l'API timer permettant le pilotage de la verte par un signal pwm, comme pour la led rouge, mais avec un rapport cyclique complémentaire.
c) ajouter la gestion des boutons haut et bas permettant de prendre en compte un appui maintenu pour augmenter (bouton haut) ou diminuer (bouton bas) le rapport cyclique "dutycycle".
d) Tester le programme (vérifier que le paramètre MAIN3 est défini).