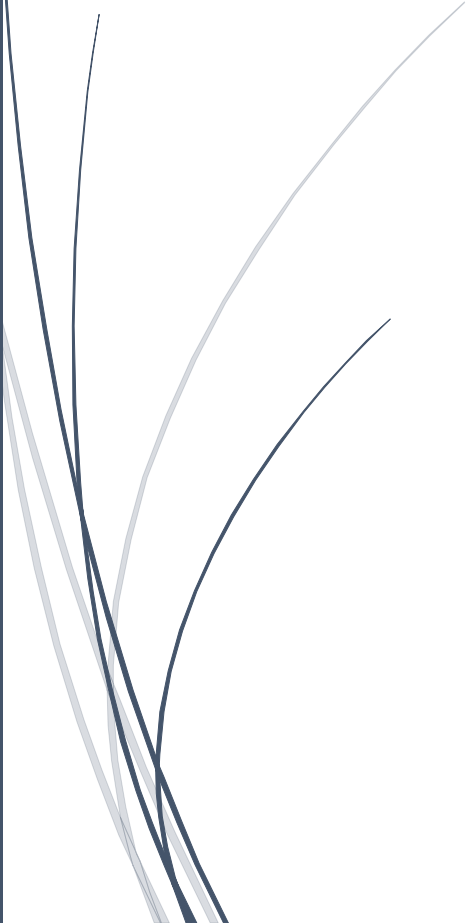
A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

4/24/2017

ASP labo 2

LCD, GPIO, INTERRUPT et Timers : le jeu du temps de réaction

Several thin, curved lines in dark blue and light grey originate from the bottom left and sweep upwards and to the right.

Camilo Pineda Serna et Ludovic Richard
ASP 2017

Table of Contents

Introduction.....	2
Etape 1 : écran LCD.....	2
Initialisation	2
Remise à zéro	2
Fonctions de la librairie lcd_toolbox.c	2
Tests	2
Étape 2 : GPIO.....	3
Initialisation	3
La gpio_toolbox.c	3
Tests	3
Étape 3 : interruption par GPIO.....	4
Initialisation	4
Routine d'interruption int_arm.S.....	4
Comportement de l'interruption	4
Étape 4 : Timer	4
Initialisation	5
La timer_toolbox.c.....	5
Etape 5 : le jeu	5
Démarrage du jeu.....	5
Terminer le jeu	5
Tests	5
Conclusion	7

Introduction

Ce rapport du laboratoire d'ASP accompagne les fichiers sources, comportant des annotations. Nous commentons étape par étape le laboratoire. Ce dernier concerne la gestion de l'affichage de l'écran de la carte REPTAR ainsi que l'implémentation de quelques GPIO (boutons et leds). Ensuite nous mettons en place une interruption provoquée par un des boutons et la gestion d'un timer. Enfin, tous ces éléments mise en commun nous permettent d'implémenter un jeu de mesure du temps de réaction.

Etape 1 : écran LCD

Cette partie concerne la réalisation d'une librairie de fonctions d'affichage sur l'écran LCD. Le but sera de pouvoir « écrire » pixel par pixel l'écran, ainsi que l'affichage d'une chaîne de caractères comme `printf`.

Avant de pouvoir écrire des pixels ou d'afficher des chaînes de caractères, l'écran doit être initialisé et remis à zéro.

Initialisation

L'initialisation de l'écran étant déjà mise en place, nous avons uniquement appelé les fonctions `lcd_off()`, `lcd_init()`, `lcd_on()` dans `main.c/general_init()`.

Remise à zéro

Notre fonction de remise à zéro de l'écran, `lcd_toolbox.c/clear_screen()`, parcourt tous les pixels de l'écran et leur assigne la couleur de fond `lcd_bg_color`. Notez le décalage de 2 octets en deux octets puisque chaque pixel est encodé sur 16 bits.

Fonctions de la librairie `lcd_toolbox.c`

Nous avons implémenté les fonctions demandées `get_pixel_add()`, `fb_set_pixel()`, `fb_print_char()` et `fb_print_string()`. Voici quelques commentaires concernant nos implémentations.

- Les pixels étant sur 2 octets, le parcours de leurs adresses se font de 2 en 2.
- La fonction d'affichage des caractères est régie par les données dans `fb_font_data[]`, représentant quels pixels sont allumés ou non. Nous parcourons ces données et « colorions » de la couleur d'arrière-plan les pixels « éteints » et avec la couleur d'avant-plan les pixels devant être allumés.
- La fonction `fb_print_string()` comporte une gestion rudimentaire de retour à la ligne et le texte retourne en début de ligne suivante, pour éviter de perdre du texte par le bord droit de l'écran.

Tests

Voici une capture d'écran démontrant la première partie de l'écran LCD.

La figure ci-dessous montre l'affichage voulu de caractères et d'une chaîne de caractère avec le retour à la ligne et le choix du comportement de `\r`. Ceci est une amélioration mineure de simplement écrire des pixels et risquer d'en perdre si leur position devait se trouver en dehors du bord droit. Notez qu'aucune gestion du bord bas n'a été faite (nous avons préféré passer aux GPIO). Une possibilité aurait été d'avoir un tampon circulaire pour que le texte qui déborde par le bas se retrouve en première ligne et réécrive l'écran.

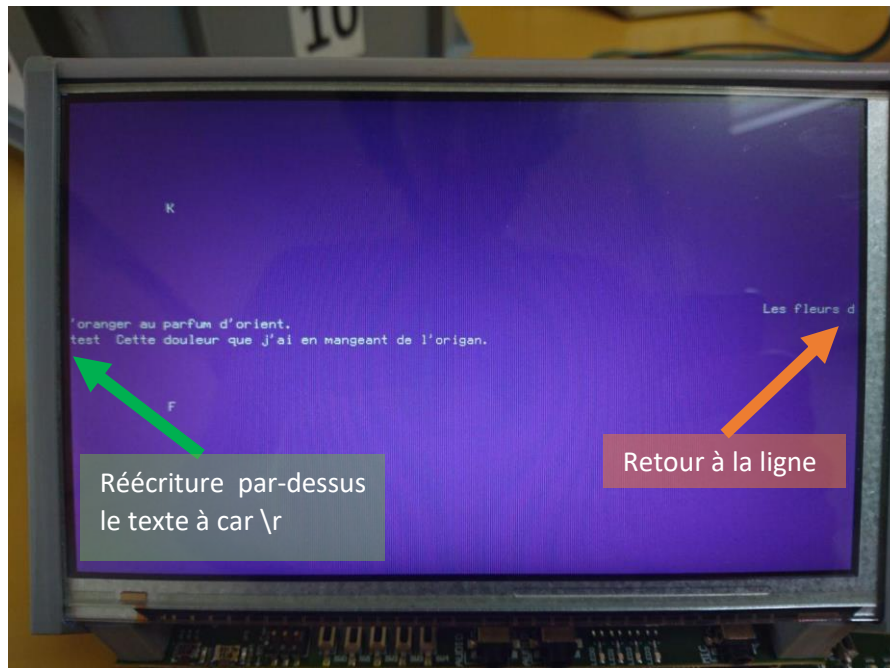


Figure 1: Première étape. Nous avons affiché 2 caractères achiraux (K et F) pour tester `fb_print_char()` et les positions passées en paramètre. Nous avons ensuite utilisé `fb_print_string()` pour tester l'affichage, le retour à la ligne (en orange) et la réécriture par-dessus les caractères déjà inscrits (en vert). Le texte original passé à la fonction était : "Les fleurs d'oranger au parfum d'orient.\n\t\tCette douleur que j'ai en mangeant de l'origan.\rtest".

Étape 2 : GPIO

Cette partie concerne les boutons (switches, SW) de la carte. Le but est d'écrire les fonctions d'initialisation et d'accès aux boutons et leds (les entrées et sorties).

Initialisation

La fonction d'initialisation `init.c/GPIO_init()` effectue un software reset puis va configurer les `PadConfValLED/SW`. Ces valeurs permettent, grâce à un mode (ici 4), de configurer le type de GPIO (présence de résistances de pull_down ou up, par exemple). Ensuite nous avons indiqué s'il s'agissait d'une entrée (les boutons) ou d'une sortie (les leds).

La `gpio_toolbox.c`

Nous avons implémenté des fonctions pour allumer, éteindre, inverser et lire l'état des leds (les fonctions `SetLed(..)`, `ResetLed()`, `ToggleLed(..)` et `ReadLed(..)`), ainsi que des méthodes pour lire l'état des boutons (`SWIsPressed(..)`). Ces fonctions vont utiliser les registres des GPIO (ici le `GPIO5_REG`).

Tests

Nous avons testé la pression d'un bouton commandant l'éclairage d'une led. Cette dernière est allumée tant que le bouton est pressé. Voici une capture d'écran de démonstration.

La figure ci-dessous illustre le comportement voulu lors de la pression sur le bouton : la led s'allume. Lorsque le bouton est relâché, la led s'éteint.



Figure 2 : étape 2. À gauche : Lorsque le bouton n'est pas pressé, la led est éteinte. À droite : Lorsque le bouton est pressé, la led est allumée. Ce comportement est géré dans la boucle active du main (après l'initialisation et l'affichage) et est constamment vérifié (attente active, pas encore d'interruption).

Étape 3 : interruption par GPIO

Cette partie concerne l'interruption par un bouton. Cette interruption (la pression du bouton) va inverser l'état d'une led. De nouveau, une initialisation des interruptions doit être réalisée avant de pouvoir les employer. L'interruption est capturée et une routine d'interruption (`int_arm.S`) appelle la fonction décrivant le comportement à adopter en allant aux instructions de `init.c/isr_handler()`.

Initialisation

La fonction d'initialisation `init.c/interrupt_init()`, après un software reset, va sauvegarder l'adresse de la routine d'interruption qui sera chargée dans le PC en cas d'interruption. Les registres sont configurés et les interruptions sont démasquées. Enfin, le bit d'interruption du CPSR est remis à zéro.

Routine d'interruption `int_arm.S`

En plus de sauvegarder puis restaurer le contexte des registres, cette routine ne fait « que » appeler la fonction `init.c/isr_handler()`.

Comportement de l'interruption

Selon l'étape en cours, le comportement de l'interruption est différent. Lors de cette étape, l'état d'une led est inversé.

Étape 4 : Timer

Cette partie concerne la mise en service d'un timer pour mesurer le temps écoulé entre l'allumage d'une led et la pression sur un bouton. L'utilisation de timers doit être configurée et activée. Le but est d'écrire une librairie pour manipuler le timer afin de pouvoir le démarrer et l'arrêter, ainsi que de lire et d'écrire sa valeur (pour le remettre à zéro). De plus, des fonctions de conversion entre la fréquence du timer et des ms furent implémentées.

Initialisation

L'initialisation se fait dans `init.c/timer_init()`. Après un `softReset`, l'on sélectionne l'horloge de source (ici une horloge env. 32kHz), puis l'on active une horloge fonctionnelle comme timer et son interface pour qu'elle soit accessible par les différents modules.

La `timer_toolbox.c`

Les fonctions de gestion du timer sont des écritures et des lectures dans le registre du timer. La conversion entre la valeur du timer et les millisecondes consiste à ajuster par un facteur 32 (puisque l'horloge a une fréquence de 32khz).

Etape 5 : le jeu

Cette partie concerne l'implémentation d'un jeu de mesure du temps réaction. Le but est d'appuyer sur un bouton le plus rapidement possible après qu'une led se soit allumée. Nous avons implémenté la fonction `application.c/jeuDeReaction()` pour l'initialisation et le démarrage du jeu. Cette fonction gère également l'affichage « entre les parties » du jeu (le meilleur score, la valeur du timer). Une fois le jeu démarré, la gestion du bouton pour terminer le jeu est gérée par interruption dans la fonction `isr_handler()`. Cette dernière comporte la logique de calcul du temps de réaction et de l'affichage « in game » du temps de réaction et du score actuel. Dans cette fonction se trouve également la gestion d'une pression précoce sur le bouton de fin de jeu.

Démarrage du jeu

Le programme attend activement (polling) la pression du bouton SW1 pour démarrer le jeu. Ceci implique que chaque pression de ce bouton redémarre le jeu. Il n'y a pas de « protection » contre ce reset. Cette attente se fait dans `application/jeuDeReaction()`, dans une boucle d'attente active. Nous avons trois états du jeu : initialisation (`init`), en cours de partie (`running`) et en attente de pression (`you can press`). Cet état permettra à l'interruption de savoir quel cas de figure est en cours (pression précoce, pression correcte, jeu pas encore démarré).

Terminer le jeu

La pression du bouton SW0 est gérée par interruption. Dès qu'une pression est effectuée, la routine d'interruption `init.c/isr_handler()` « partie Pour étape 5 » s'exécute. Selon l'état du jeu (voir plus haut « Démarrage du jeu »), la routine va soit rien faire (pas de jeu en cours), soit afficher « cheater » et effacer le meilleur score (pression précoce), soit calculer le temps de réaction et l'afficher (pression correcte). Si un nouveau meilleur score est établi, son affichage remplace le précédent. L'état du jeu retourne alors à celui d'initialisation.

Tests

Voici des images de gameplay de notre jeu.

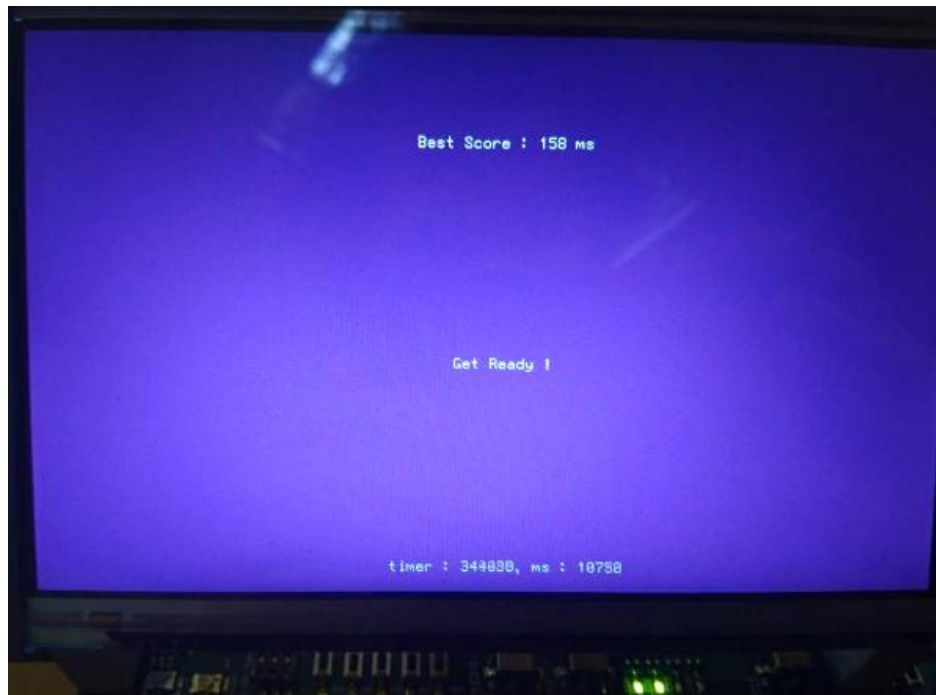


Figure 3 : Le jeu est en cours et la led marquant le début de l'état « you can press » est déjà allumée (celle de gauche). Pour terminer le jeu, il manque la pression sur le SW0. Sur l'écran, de haut en bas : Le meilleur score courant, le message de début de jeu et les valeurs du timer, également en millisecondes.



Figure 4 : jeu terminé et inscription d'un nouveau meilleur score. Au centre de l'écran se trouvent les messages avec la valeur du temps de réaction en millisecondes de la partie venant de se terminer, ainsi qu'un petit message d'appréciation.

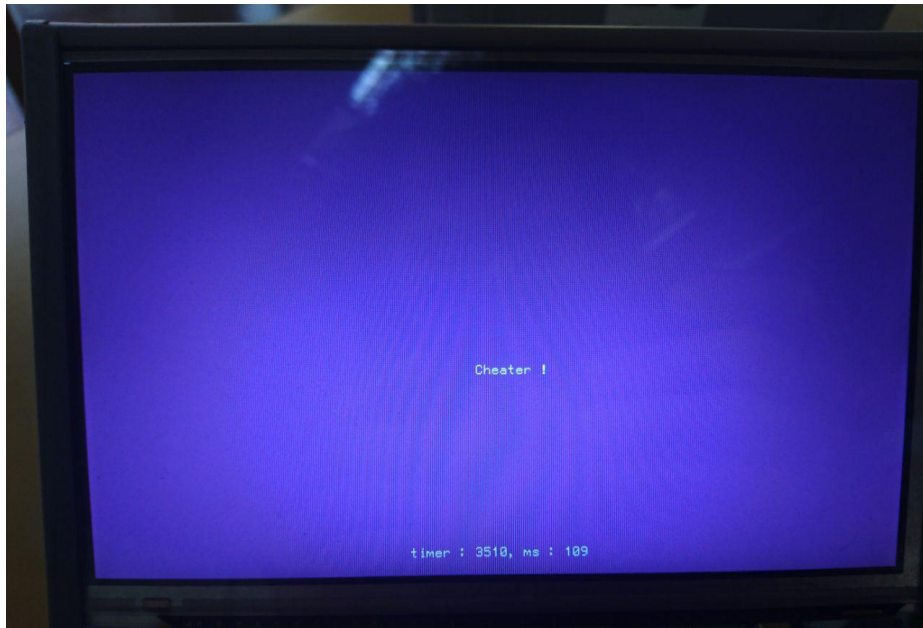


Figure 5 : écran en cas de pression précoce sur le bouton. Le meilleur score est effacé (plus de texte en haut) et un message s'affiche au centre de l'image.

Conclusion

En plus d'implémenter des fonctionnements d'un niveau plutôt proche de la machine, nous avons dû naviguer et nous familiariser avec une documentation officielle d'un composant complexe. Une partie non négligeable du temps passer fut de retrouver quels registres modifier et avec quelles valeurs dans cet immense document. Nous avons ainsi aperçu à quel point, même pour des fonctionnements plutôt « basiques » (comme afficher une chaîne de caractères ou allumer une led), les plateformes et composants modernes doivent initialiser nombre de paramètres. Nous apprécions ainsi que ce soit déjà fait et fait correctement dans les ordinateurs et autres smartphones que nous utilisons quotidiennement.