Architecture des Systèmes à Processeur

R. Mosqueron / A. Convers

Labo Périphériques

28.02.2017 – V10

Informations générales

Le labo Périphériques s'effectue durant 12 périodes encadrées.

Vous devrez rendre un rapport global plus le code. Code + rapport à rendre le **09.04.2017 (23h59) dernier délai**.

Le rendu s'effectue sous forme de :

a) Version informatique du code : envoyer les répertoires des deux projets compressés dans une archive à anthony.convers@heig-vd.ch (faire un make clean avant pour n'inclure que les sources)

b) Version informatique du rapport : fichier .pdf à envoyer à anthony.convers@heig-vd.ch

Objectif du laboratoire

L'objectif de ce laboratoire est d'écrire et tester les drivers et fonctions de bas niveau qui permettront d'initialiser votre carte et de rendre opérationnels quelques périphériques : afficheur LCD, boutons poussoir, leds, timers. Ce laboratoire se terminera par la réalisation d'un jeu.

Ce laboratoire vous permettra de comprendre et de mettre en œuvre les techniques d'interfaçage des boutons poussoir et des leds par GPIO, les mécanismes de traitement des interruptions, la mise en œuvre des timers et l'interfaçage de l'écran LCD. Il s'agira de configurer les différents registres du DM3730 nécessaires à l'initialisation des GPIO, des interruptions et des timers, d'implémenter une Interrupt Service Routine (ISR) et des fonctions d'affichage LCD.

Rappelons que pour ce laboratoire comme pour les suivants d'ASP, le programme écrit devra fonctionner en stand-alone sans utiliser de code déjà installé dans la carte, autrement dit : ni moniteur ni OS. Les drivers et fonctions réalisés seront utilisés dans les prochains labos.

Ce laboratoire est noté. Vous devez rendre le code écrit **commenté** et un rapport. Le rapport doit montrer votre compréhension des fonctions hardwares du microcontrôleur, et expliquer comment votre code inter-réagit avec le hardware (écriture et lecture de registres). Ce rapport doit comprendre aussi une brève partie journal qui montrera votre progression au cours du labo et les problèmes auxquels vous avez pu être confrontés.

Fichiers et documents fournis

Vous trouverez les informations nécessaires à la réalisation des différentes étapes de ce laboratoire dans les documents ci-dessous:

- Technical Reference Manual du DM3730, pus particulérement :
  - chapitre 25 - GPIO
  - chapitre 13 – SCM (Pad configuration)
  - chapitre 12 – INTC (Interrupt Controller)
  - chapitre 26 – Initialization (Seulement 26.4.2.2 RAM Memory Map)
  - chapitre 16 – Timers
  - chapitre 3 – PRCM (Power, Reset and Clock Management, Seulement 3.5.3 Clock Manager, 3.6 Basic Programming Model et 3.7 Register Manual)

Mettez à jour votre répertoire Git pour obtenir les nouveaux fichiers.

username@eigit:/home2/reds/asp/asp_student

Le dossier « labo_ASP_perif » contient les fichiers suivants (voir informations ci-dessous) :

init.c, lcd_toolbox.c

bits.h, gpio.h, lcd.h, fb_fonts.h, vga_lookup.h, init.h, padconf.h, lcd_toolbox.h, gpio_toolbox.h prcm.h, timer.h, intc.h

Structure des fichiers pour le labo Périphériques:

Fichiers .c

main.c

main()	appel de la fonction general_init et de la fonction qui lance l'application spécifique (par exemple le jeu temps de réaction)

init.c

xxx_init()	fonction d'initialisation de chaque module (GPIO, timer, LCD…)
interrupt_init()	fonction d'initialisation des interruptions, **à appeler en dernier**
isr()	routine de service d'interruption
general_init()	appel des diverses fonctions _init

lcd_toolbox.c

xxxxx()	*vos fonctions d'écriture LCD*

gpio_toolbox.c

xxxxx()	vos fonctions d'accès boutons et leds

timer_toolbox.c

xxxxx()	vos fonctions d'accès au timer

applications.c
- temps_reaction() fonction jeu temps de réaction
- xxxxx() *vos applications*

Fichiers .h

cfg.h	taille de la pile
bits.h	définitions pour accès bit par bit (masques)
stddefs.h	définitions de types
gpio.h	définitions pour configuration des registres GPIO du DM3730
lcd.h	définitions pour configuration des registres DSS du DM3730
timer.h	définitions pour configuration des registres du module Timers du DM3730
prcm.h	définitions pour configuration des registres PRCM du DM3730
padconf.h	définitions pour configuration des registres SCM du DM3730
intc.h	définitions pour configuration des registres INTC du DM3730
fb_fonts.h	définition police bitmap de caractère
vga_lookup.h	définitions de couleurs standards pour le LCD
init.h	définitions et prototypes initialisation de base, interruptions
lcd_toolbox.h	définitions pour fonctions d'accès au LCD
gpio_toolbox.h	*vos définitions pour fonctions d'accès aux GPIOs*
timer_toolbox.h	*vos définitions pour fonctions d'accès au timer*
applications.h	vos *définitions et prototypes pour applications*

Fichiers .S

crt0_arm.S	fichier assembleur de démarrage, initialisation de la pile

Fichiers divers

Makefile	paramètres de compilation et linkage
standalone.ld	linker script, décrit l'organisation en mémoire des sections du fichier exécutable .elf
newlib_stubs.c	fonctions bas niveau que doivent être fournies par un système afin de rendre utilisables des librairies standard, notamment stdio et stdlib

Travail à effectuer

AVANT DE COMMENCER

1- Depuis un explorateur de fichiers ou en ligne de commandes créer un nouveau répertoire de projet « **labo_ASP_perif** » dans votre workspace. Copiez vos fichiers du labo précédent. N'ajoutez pas les nouveaux fichiers. Lancez Code Composer Studio et suivez le tutoriel « Utilisation de CCSv5 » pour créer un nouveau projet. Vérifiez la compilation et le chargement dans la carte.

2- Ajoutez les nouveaux fichiers. Modifiez le Makefile afin que les nouveaux modules *.c* soient pris en compte lors de la compilation et modifiez le fichier main.c pour ajouter les directives #include correspondant aux *.h* de chaque module *.c*. Vérifier la compilation et le chargement.

Etape 1 : Ecran LCD

Vous devez réaliser une librairie de fonctions lcd_toolbox d'affichage sur l'écran LCD :
- *Ecriture pixel par pixel*
- *Ecriture de chaîne de caractères (équivalent printf)*

Informations :

- Chaque pixel est représenté avec 2 octets. Le format de l'écran est 480 x 800. On utilise le code RGB 565 (5 bits pour le rouge, 6 bits pour le vert et 5 bits pour le bleu). Le tableau **vga_lookup[ ]** décrit dans **vga_lookup.h** définit 16 couleurs en utilisant le code mentionné.
- Pour allumer un pixel de l'écran à une position déterminée, il faut écrire les 2 octets correspondant à la couleur souhaitée dans un buffer en RAM (Frame Buffer). L'adresse du Frame Buffer est définie dans **lcd.h** et sa taille en octets se calcule ainsi Tmax = 480 x 800 x 2. L'adresse d'un pixel dans le Frame Buffer se calcule à partir du coin haut gauche ligne par ligne (taille ligne = 800 pixels).
- La macro **LCD_BUF(_x_)** définie dans **lcd.h** donne accès au contenu du Frame Buffer à l'adresse **_x_**
- Police de caractère bitmap : Le tableau bidimensionnel **fb_font_data** défini dans **fb_fonts.h** représente un caractère ASCII par ligne. Chaque caractère est décrit par un rectangle de 8x8 pixels ou de 8x16, où chaque pixel est représenté par un bit. La taille de caractère est définie par la constante USE_FONTxxx

*Ecrire les fonctions d'affichage sur le LCD dans le fichier **lcd_toolbox.c** (avec les déclarations de fonctions dans lcd_toolbox.h).*
Ecrivez une fonction de test dans le fichier main (écriture de quelques lignes de texte dans un rectangle dessiné, par exemple).

1. Ecrire la fonction d'initialisation de la carte **general_init()** dans le fichier main.c, elle doit appeler dans l'ordre, **lcd_off(), lcd_init(), lcd_on()**
2. Ecrire la fonction **clear_screen** qui efface l'écran.
3. Ecrire la fonction **get_pixel_add** qui rend l'adresse d'un pixel à partir de ses coordonnées xy sur l'écran.
4. Ecrire la fonction **fb_set_pixel** pour allumer un pixel de la couleur souhaitée à une position xy passée en paramètre.

5. Ecrire la fonction **fb_print_char** qui affiche un caractère de la couleur souhaitée à une position passée en paramètre.
6. Ecrire la fonction **fb_print_string** qui affiche une chaine de caractères de la couleur souhaitée à une position passée en paramètre.
7. Tester et déboguer toutes les fonctions en les appelant depuis le main. N'oubliez pas d'appeler en premier la fonction d'initialisation **general_init().**
8. Facultativement, pour un usage ultérieur, vous pouvez écrire des fonctions supplémentaires : Traçage de ligne, de rectangle, de fenêtre (avec texte)

Etape 2 : Initialisation des GPIOs

Note : Les boutons et les leds à utiliser sont ceux de la carte REPTAR CPU, voir figure 2, références 5 et 8 dans le document sur CCS.

Ecrire les fonctions d'initialisation et d'accès aux boutons et aux leds dans le fichier ***gpio_toolbox.c*** *(avec les déclarations de fonctions dans gpio_toolbox.h).*

1. Lire le chapitre 25 (GPIO) du manuel du DM3730
2. Les numéros de GPIO correspondant aux boutons poussoir et LEDs de la carte CPU sont donnés ci-dessous

VAR SOM pin	DM3730 GPIO_	DM3730 module/bit	Schematic reference	Board reference	Component description
PUSH-BUTTONS					
65	140	GPIO5/ bit 12	SW1	SW0	Push button
69	142	GPIO5/ bit 14	SW2	SW1	Push button
57	167	GPIO6/ bit 7	SW3	SW2	Push button
59	97	GPIO4/ bit 1	SW4	SW3	Push button
62	126	GPIO4/ bit 30	SW5	SW4	Push button
LEDs					
71	143	GPIO5/bit 15	D2	LED0	Green LED
67	141	GPIO5/ bit 13	D3	LED1	Green LED
56	101	GPIO4/ bit 5	D11	LED2	Green LED
55	102	GPIO4/ bit 6	D14	LED3	Green LED

3. Compléter la fonction **GPIO_init()** du fichier **init.c** en initialisant les registres définis dans **padconf.h** et dans **gpio.h** pour chaque pin (deux LEDs 0 et 1 ; deux boutons SW 0 et 1). Voir exemple dans fonction **lcd_init()** du fichier **init.c**
4. Créer un fichier gpio_toolbox.c pour contenir vos fonctions dont les prototypes se trouvent dans gpio_toolbox.h
    a. Ecrire les fonctions nécessaires pour allumer, éteindre et inverser l'état d'une led (*toggle*)
    b. Ecrire une fonction pour lire l'état d'un switch

5. Tester et déboguer vos fonctions

Etape 3 : Interruption par GPIOs

Le programme à réaliser a pour fonction de détecter l'appui sur le bouton SW0 et d'allumer/éteindre la led LED0 en utilisant le mécanisme d'interruption (allumage avec un appui, extinction avec un second appui).

*Ecrire la routine d'interruption et l'initialisation des interruptions dans le fichier **init.c**.*

Informations :

Les interruptions doivent être initialisées dans la fonction **interrupt_init** du fichier **init.c.**

Afin de générer une interruption sur appui d'un bouton, vous devez créer une routine d'interruption (**isr**) en assembleur (ajoutez un fichier .s pour cette fonction et modifiez le makefile).

Ajoutez l'adresse de cette routine dans la table des vecteurs. Cette fonction doit appeler une fonction handler d'interruption (**isr_handler**) à placer dans **init.c**.

Vous compléterez le handler d'interruption avec les tâches que vous sembleront utiles.

Note :
Utilisez les *defines* des divers fichiers .h donnés pour accéder aux registres.

1- Lire tous les chapitres du DM3730 listés auparavant (sauf chapitre 16).
2- Ecrire les fonctions suivantes :
   - **interrupt_init** dans le fichier **init.c**.
   - **isr_handler** dans le fichier **init.c**.
   - **isr** en assembleur dans le fichier **int_arm.s** que vous créerez.
3- Testez le programme sous Code Composer Studio.

Etape 4 : Mise en œuvre d'un timer

Le programme à réaliser a pour fonction de mesurer le temps écoulé entre l'allumage d'une led et l'appui sur un bouton à l'aide d'un timer.

1. Lire le chapitre 16 (Timers) du manuel du DM3730.
2. Ecrire la fonction d'initialisation **timer_init** dans le fichier **init.c**. Elle doit initialiser le timer GPTIMER1. Pour initialiser le timer, vous devez faire un Reset software du timer et ensuite sélectionnez l'horloge source du timer. Pour ce dernier point configurer les registres CM_CLKSEL, CM_FCLKEN et CM_ICLKEN du module PRCM (defines dans **prcm.h**).
3. Créer les fichiers **timer_toolbox.c** et **timer_toolbox.h** et y écrire les fonctions d'accès au timer suivantes: **start_timer**, **stop_timer**, **read_timer_value** et **write_timer_value**.
4. Le programme doit déclencher un timer à l'allumage de la led. A l'arrivée de l'interruption, la valeur du timer sera stockée dans une variable globale.
5. Testez le programme sous Code Composer Studio et visualisez la valeur de la variable (dans la fenêtre « expressions » de la perspective **Debug**).

Etape 5 : Application : Jeu mesure temps de réaction

Vous devez réaliser le programme d'un jeu « mesure du temps de réaction » : Le joueur presse un bouton start (SW1), une led s'allume après un temps aléatoire (LED0), le joueur appuie le plus vite possible sur un bouton (SW0) dès que la led s'allume et le temps de réaction en ms s'affiche sur l'écran LCD.

L'appui sur le bouton SW1 sera détecté par polling et l'appui sur le bouton SW0 par interruption.

*Ecrire le programme dans le fichier **applications.c** (avec les déclarations dans applications.h). Ce programme sera appelé à partir du **main**.*

1. Concevoir une interface utilisateur simple. Ecrire les fonctions.
2. Testez le programme sous Code Composer Studio.

<u>Si vous avez du temps</u>, quelques possibilités de développements supplémentaires (qui vous apporteront un bonus de points).

- Affichage du meilleur score
- Plusieurs tailles de caractères sur l'écran
- Gestion de fenêtres sur l'écran
- Amélioration du jeu => utilisation des deux boutons en demandant par exemple de rentrer une séquence aléatoire fournie le plus rapidement possible.
- Si vous avez une autre idée, proposez-la au professeur.

N'oubliez pas de sauvegarder votre répertoire workspace qui vous sera nécessaire pour le prochain labo. Le répertoire étudiant est effacé à chaque extinction de machine.