h e p i a

Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

Laboratoire de systèmes numériques
D. Bechevet, A. Lescourt, V. Pilloux, F. Vannel

General Purpose Input Output (GPIO)

Labo PRO

Introduction

Ce laboratoire va vous permettre d'utiliser le périphérique GPIO et contrôler des entrées et sorties de la carte d'extension **myLab2**. Vous apprendrez à contrôler les LEDs (Diodes Electro-Luminescentes), ainsi que les interrupteurs présents sur la carte. Vous écrirez également plusieurs des programmes de ces exercices en langage C.

1 Notions d'assembleur

Pour ce laboratoire vous aurez besoin d'utiliser les instructions en assembleur décrites en cours ainsi que dans les précédents laboratoires.

2 Informations sur le matériel de laboratoire

La carte **LPCXpresso** ne possédant qu'une seul LED, vous allez devoir la brancher sur une carte d'extension **myLab2**. Cette dernière ne dispose pas de processeur, mais ajoute des interfaces complémentaires au processeur situé sur la carte **LPCXpresso**. Attention lorsque vous connectez et déconnectez la carte **LPCXpresso** à la carte **myLab2**. Elles sont **FRAGILES** et doivent être manipulées avec précaution.

3 Périphérique GPIO interne au microcontrôleur

Le périphérique GPIO (ou port d'entrées/sorties numérique) est connecté au bus d'adresse et de donnée du processeur. Il dispose d'une plage d'adresse allant de 0x2009C000 à 0x2009FFFF. Cependant, toute cette plage n'est pas utilisée et nous n'utiliserons que 25 adresses dans cet intervalle (cf. documentation du microcontrôleur). Dans un but de compréhension, des noms sont associés à ces adresses.

3.1 FIODIR

5 adresses 0x2009C000, 0x2009C020,0x2009C040, 0x2009C060 et 0x2009C080 permettent de définir la direction de chaque patte du composant. Comme cité précédemment ces adresses ont des noms associés [1] de la façon suivante :

1. Généralement appelés registres de périphériques

- FIO0DIR associé à 0x2009C000 : Permet de contrôler la direction des ports P0.0 à P0.31
- FIO1DIR associé à 0x2009C020 : Permet de contrôler la direction des ports P1.0 à P1.31
- FIO2DIR associé à 0x2009C040 : Permet de contrôler la direction des ports P2.0 à P2.31
- FIO3DIR associé à 0x2009C060 : Permet de contrôler la direction des ports P3.0 à P3.31
- FIO4DIR associé à 0x2009C080 : Permet de contrôler la direction des ports P4.0 à P4.31

Le contenu associé à ces adresses spécifie le fonctionnement de ces ports. Ainsi le bit de poids faible fait référence à la patte (ou broche) Px.0 alors que le bit de poids fort fait référence à la patte Px.31. Si le bit écrit à cet emplacement est un **0** alors la patte correspondante sera configurée en tant qu'*entrée*. Si c'est un **1** ce sera une *sortie*.

3.2 FIOPIN

Les 5 registres FIO0PIN, FIO1PIN, FIO2PIN, FIO3PIN et FIO4PIN peuvent êtres lus et également écrits.

Si la patte du composant est configurée en entrée, nous obtiendrons en lisant ces registres l'état logique de la patte.

Si la patte du composant est configurée en sortie, nous pourrons en écrivant dans ce registre affecter l'état de la patte.

Attention : une écriture sur ce registre modifie l'état de toutes les sorties du port adressé.

3.3 FIOSET

Les 5 registres FIO0SET, FIO1SET, FIO2SET, FIO3SET et FIO4SET peuvent être utilisés pour contrôler l'état des pattes configurées en sortie.

Les bits à 1 dans le registre correspondant, forceront les pattes configurées en sortie à l'état **1**. Les bits à 0 dans le registre correspondant n'auront aucune influence sur l'état des sorties.

3.4 FIOCLR

Les 5 registres FIO0CLR, FIO1CLR, FIO2CLR, FIO3CLR et FIO4CLR sont similaires à FIOSET et peuvent êtres utilisés pour contrôler l'état des pattes configurées en sortie.

Les bits à 1 dans le registre correspondant forceront les pattes configurées en sortie à l'état **0**. Les bits à 0 dans le registre correspondant n'auront aucune influence sur l'état des sorties.

3.5 FIOMASK

Les 5 registres FIO0MASK, FIO1MASK, FIO2MASK, FIO3MASK et FIO4MASK permettent de sélectionner les PINs qui seront ou ne seront pas affectées par FIOPIN, FIOSET et FIOCLR.

- 0 : le bit est accessible en lecture et en écriture.
- 1 : le bit est protégé en écriture.

Le masque filtre aussi la lecture de FIOPIN : les bits dont le mask correspondant est à 1 sont lues comme des 0. Par exemple :

```
FIO0PIN = 0xAAAAAAAA;
uint32_t a = FIO0PIN; // a = 0xAAAAAAAA
FIO0MASK = 0x0000FF00;
a = FIO0PIN; // a = 0xAAAA00AA
```

3.6 Manipulation des registres

Le fichier **config_LPC1769.h** situé dans votre projet contient les alias (ou correspondances) entre les *adresses des registres de périphériques* et le *nom associé*. Vous êtes fortement invité à aller consulter ce fichier afin de mieux comprendre le principe des registres de périphériques.

Par la suite, dans ce laboratoire vous pourrez affecter des valeurs à ces registres très simplement, par exemple :

```
FIO2SET = (1<<2);  // Force la sortie P2.2 a l'etat 1
uint32_t data;
data = FIO2PIN; // copie dans data la valeur des 32 ports P2.31 a P2.0
```

3.7 Connexion LPCXpresso - MyLab

Voir documentation spécifique **myLab2** sur `http://hepialsn.hesge.ch/myLab2`.

4 Le projet sous LPCXPresso

Vous importerez dans votre *workspace* le projet **labo4_GPIO**.

Le fichier **labo4_GPIO.c** contient la fonction **main**, l'appel aux fonctions des exercices ainsi que les fonctions C à compléter

Le fichier **assembleur.s** contient le code en assembleur que vous devrez compléter.

Pour tester votre code et le fonctionnement adéquat par rapport à votre exercice, vous devrez soit modifier la ligne suivante se trouvant dans le début du **main**, soit modifier durant l'exécution la valeur de cette variable :

```
exo=1;
```

Le numéro correspond à l'exercice que vous souhaitez tester.

5 Debugger des périphériques

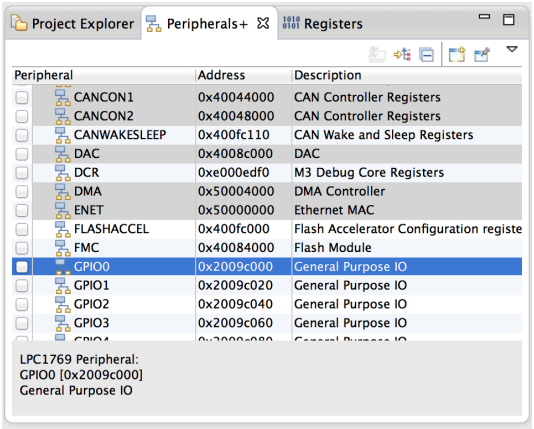

FIGURE 1. *Sélection du périphérique à debugger*

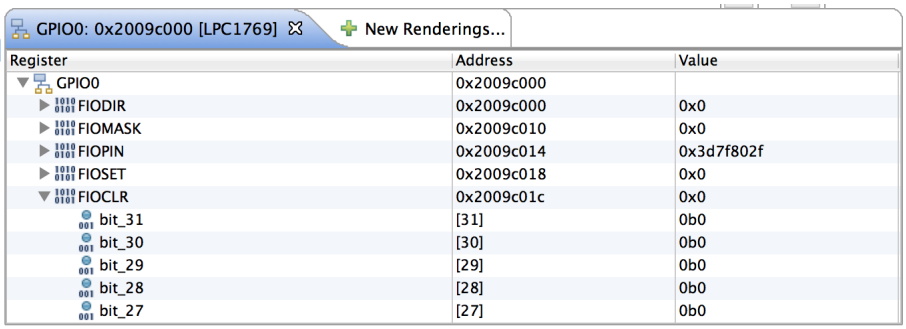

FIGURE 2. *Vue debugger du GPIO (P0) permettant de modifier et lire les registres des périphériques*

6 Exercices

6.1 Contrôle de LEDs en Assembleur

Ecrivez en Assembleur un programme effectuant la séquence lumineuse suivante de façon continue :

- LED 0 allumée.
- LED 1 allumée.
- LED 0 éteinte.
- LED 1 éteinte.

N'oubliez pas l'initialisation des sorties à effectuer une seule fois au début de cette fonction. Vous testerez votre programme en mode pas à pas uniquement.

Note : Pour charger une valeur de 32 bits dans un registre, il faut utiliser l'instruction LDR de la façon décrite ci-après. Ceci revenant à aller charger dans un registre destination le contenu d'une adresse située dans le programme (accessible par ajout d'un *offset* au registre **PC**) :

```
.equ _FIO2DIR, 0x2009C040  // adresse du register FIO2DIR
function:
  ...
  LDR R0, =_FIO2DIR       // R0 = 0x2009C040 (FIO2DIR)
  ...
```

Attention : Afin que la valeur qui sera affectée à l'aide du mot clé `.equ` ne soit pas exécutée, il faut que celle ci se trouve en dehors d'une fonction.

6.2 Initialisation des GPIO

Avant de commencer à manipuler les entrées et sorties de la carte il est important d'initialiser la direction de chaque GPIO que vous aller utiliser sur la carte **myLab2**. La fonction `void init (void);` permet de mettre le code qui devra être exécuté à chaque démarrage de la carte et contenant entre autre l'initialisation des directions et états des GPIO.

Cette fonction devra initialiser la direction pour les entrée et les sorties en fonction des besoins des exercices ci-après. Dans un premier temps commencer à compléter cette fonction selon les besoins du prochain exercice, puis modifiez le contenu selon les exercices suivants.

6.3 Affichage clignotant

Ecrivez le programme suivant en langage C. Vous allez générer la séquence lumineuse suivante de façon continue :

- **étape 1 :** LED 0 allumée. LED 7 éteinte.
- **étape 2 :** LED 0 éteinte. LED 7 allumée.

Vous effectuerez une pause d'environ 1 seconde entre chaque étape (Etant donné que nous n'avons pas encore étudié le périphérique TIMER permettant de contrôler le temps, vous réaliserez ce délai d'attente avec une boucle *for*).

6.4 Compteur 8 bits à deux vitesses

Vous déclarerez une variable que vous incrémenterez périodiquement. Les 8 bits de poids faibles de cette variable devront être affichés sur les 8 LEDs à disposition. La vitesse de ce compteur doit être contrôlée par l'état du joystick de la façon suivante :

- Joystick appuyé = délai de 0.5 seconde
- Joystick relaché = délai de 1 seconde

L'utilisation de FIOPIN en écriture n'est possible dans cet exercice qu'en utilisant FIOMASK également, sans quoi cela modifierait l'état d'autres sorties sur la carte **LPCXpresso**, ce que nous ne souhaitons pas.

6.5 Compteur 8 bits manuel

En reprenant votre travail de l'exercice précédent, vous adapterez votre code afin de ne plus avoir de délai, mais incrémenterez le compteur de 1 à chaque appui sur le joystick. (Note : Les solutions utilisant des délais ne sont pas acceptées)

6.6 Structuration du code

6.6.1 void LedSetState (uint8_t led, bool state)

Créez une procédure selon le prototypage ci-dessus permettant de définir l'état d'une LED.

- led est le numéro de la led (dans notre cas une valeur de 0 à 7).
- state est une valeur booléenne donnant l'état de la led correspondante. **FALSE** équivaut à éteint. **TRUE** à allumé.

6.6.2 void Led8SetState (uint8_t value)

Créez une procédure selon le prototypage ci-dessus permettant de définir l'état des 8 LEDs.

- value est une valeur donnant l'état des 8 LEDs correspondantes.

6.6.3 bool JoystickGetState (uint8_t pos)

Créez une fonction selon le prototypage ci-dessus permettant d'obtenir l'état du Joystick.

- pos est la position du joystick à tester et définit selon votre choix. Il y a 5 positions possible pouvant être testé sur ce joystick.
- Cette fonction renverra l'état du joystick testé, à savoir `1` s'il est en appuyé ou déplacé, `0` sinon.

Remarque : Réfléchissez afin que cette fonction soit écrite en n'utilisant aucun `SWITCH CASE`, ni `IF ELSE`.

6.6.4 uint8_t SwitchGetState (void)

Créez une fonction selon le prototypage ci-dessus permettant d'obtenir l'état des 8 interrupteurs.

6.6.5 void delai(uint16_t delai)

Créez une procédure gérant un délai selon le prototypage ci-dessus.

- Cette fonction est une attente active. Le `délai` (approximatif) est exprimé en milliseconde.

6.7 Chenillard lumineux

En s'assurant que seule la LED 0 de la carte **myLab2** est allumée au démarrage, vous réaliserez un chenillard lumineux effectuant, chaque seconde, le décalage de la LED allumée de droite à gauche. Une fois la LED 7 allumée, la suivante sera de nouveau la LED 0.

L'interrupteur I.0 contrôle le sens du chenillard :

- ON : déplacement de droite à gauche
- OFF : déplacement de gauche à droite

Utilisez les fonctions écrites précédemment.

6.8 Chenillard avec motif rebondissant

Dans cet exercice il vous est demandé de modifier votre chenillard afin que celui-ci change automatiquement de sens dès que la LED d'une extrémité est allumée.

Les 8 interrupteurs devront être utilisés pour créer un motif à charger dans le chenillard.

A chaque appuis sur le joystick, le motif dessiné sur les interrupteursdoit être chargé sur les LEDs et le chenillard devra commencer dans le sens droite vers gauche.

Si le motif sur les interrupteurs n'a pas été modifié et que le bouton du joystick est tout de même pressé, le sens du chenillard devra tout simplement changer de sens.

Utilisez les fonctions précédemment créées.

6.8.1 Librairie C

Afin d'avoir un programme clair, vous allez créer une librairie dans laquelle vous allez déplacer les différentes fonctions précédemment créées.

Dans le projet que vous avez importé lors du **labo1**, un projet nommé **MyLab_lib** a été ajouté avec différents fichiers.

Ajoutez les fichiers suivants

- **MyLab_gpio.c** Ce fichier devra contenir toutes les fonctions contrôlant les interfaces LEDs, joystick et interrupteurs.
- **MyLab_gpio.h** Ce fichier devra contenir toutes les prototypes des fonctions qui se retrouvent dans le fichier **MyLab_gpio.c**.

Vous pouvez ajouter des fichiers similaires pour gérer d'autre fonctionnalités. Par exemple des fichiers **utils.c** et **utils.h** permettant de gérer les délais, etc... Ces librairies seront complétées durant les prochains laboratoires.

6.9 Utilisation de la librairie

Modifiez vos exercices de ce laboratoire afin d'utiliser les nouvelles fonctions de vos librairies. Pour ceci, il faut `inclure` les fichiers contenant les prototypes en début de programme.

Rappel : COMMENTEZ VOTRE CODE.
Bon travail.