ÉCOLE POLYTECH'NICE SOPHIA
SYSTÈMES ET APPLICATIONS EMBARQUÉS
2013-2014

Julien DeAntoni

Systèmes embarqués

Programmation micro-contrôleur

Sans OS

Introduction

Comme précisé dans le cours, nous allons programmer un micro-contrôleur ATMEL dont le petit nom est atmega328p. Ce micro-contrôleur est relativement complet mais nous n'expérimenterons dans ce TP qu'une partie de ses possibilités.
La première étape va être de récupérer les *datasheets* sur internet (par xemple dans le zip ici : http ://www-sop.inria.fr/aoste/personnel/Julien.Deantoni/enseignements/iam03/sceance_1/). Mais attention, ne vous amusez pas à les lire en entier sinon vous n'aurez jamais le temps de faire le moindre programme. Par contre vous pouvez lire la première page pour vous mettre dans l'ambiance...
Puisque nous avons peu de temps alloué, nous allons faire des programmes "par l'exemple". Ainsi, après avoir installé la suite logicielle de votre choix vous testerez les exemples fournis dans le zip sur ma page internet. Ensuite vous pourrez les modifiez pour atteindre les objectifs donnés dans la suite.
Un résumé des fonctions des pattes du micro controlleur est donné figure 1

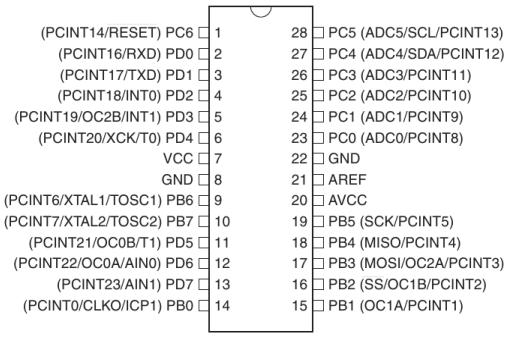

FIGURE 1 – Résumé des fonctions des pattes du micro controlleur atmega328p

1 Premiers pas

1.1 Installation de la suite logicielle

Sous Linux (de préférence sinon débrouillez vous :-/), installez maintenant la suite logicielle suivante :

— compilateur / simulateur
  — avr-gcc : sous linux, les packages existent pour la majoritée des distributions (sous le nom gcc-avr). `apt-get, yum`, etc sont vos amis.
  — avr-libc : similaire au point précédent.
— *linker*
  — avrdude : devrait être dans les packages des distributions linux.

1.2 Installation matérielle

Durant ces exercices, vous allez utiliser la carte arduino (bien sûr...) et une carte permettant de relier divers capteurs/actionneurs. L'ensemble est alimenté par un cable USB. C'est également ce cable qui permettra de transférer votre programme sur le micro controlleur.
Vous prendrez soin de ne pas laisser trainer d'objets métalliques sur votre table (afin d'éviter de court-circuiter les cartes et éventuellement votre port USB). Vous ferai également attention à ne pas laisser les capteurs/actionneurs faire des court-circuits.

1.3 Hello World sur la carte arduino

Dans le monde de la programmation micro-contrôleur il n'est pas rare de ne pas avoir d'écran. De plus, lorsque l'on en a un, c'est alors un programme complexe qu'il faut mettre en oeuvre pour l'utiliser. De ce fait, l'équivalement du traditionnel "*Hello World*" est "*LED Blink*". Ce dernier, comme vous l'avez tous compris consiste à faire clignoter une diode éléctro-luminescente (il est effectivement rare de trouver une carte micro-contrôleur sans aucune LED). C'est donc le programme qui vous est fournis pour vos premiers tests. La led se trouve sur la carte arduino. Elle est reliée au bit 5 du port B (PB5).
Téléchargez le sur ma page (ou regardez dans le zip) ainsi que le Makefile associé.

compiler led_blink.c : Assez classiquement faites un `make` dans un terminal. Attention, lorsque vous réutiliserez ce Makefile, le nom du projet est en dur au début du fichier, il correspond au nom du fichier .c principal.

transférer dans le micro-contrôleur : Le makefile possède une cible nommée *upload* qui utilise avrdude pour transférer le programme compilé vers la carte. Cette ligne suppose que la carte a été reconnue come `/dev/ttyACM0` par votre OS. Si ce n'est pas le cas, ajustez...
Vérifiez que la la led orange de la carte clignote...
Maintenant regardez votre code, comprenez le et posez vous quelques questions. Par exemple :
— Comment se fait-il que l'on puisse utiliser directement les noms des ports et que l'on ne soit pas obligé de donner leur adresse ?
— Que se passe t'il si l'on ajoute dans le Makefile l'option '`-O3`' qui permet de mettre en oeuvre les optimisations de gcc ? Testez et expliquez.
— ... ?

2 Premier (deuxième ?) programme (sans interruptions)

Vous allez maintenant ajouter la carte d'interfacage d'entrée sortie sur la carte arduino (faites ca correctement sans rien casser !). Sur cette carte, vous allez branchez les deux leds fournies dans le kit (stater kit) sur les bornes `D8` et `D11`. Ces bornes sont respectivement liées à `PB0` et `PB3`. La figure suivante (figure 2) montre les lien entre les ports du micro controlleur et les bornes de la carte d'interfacage rapide.
Vous brancherez le bouton poussoir sur la borne `A1`, reliée au à la patte `PC1`. (Il vous sera donc nécessaire de mettre le port C en entrée.)

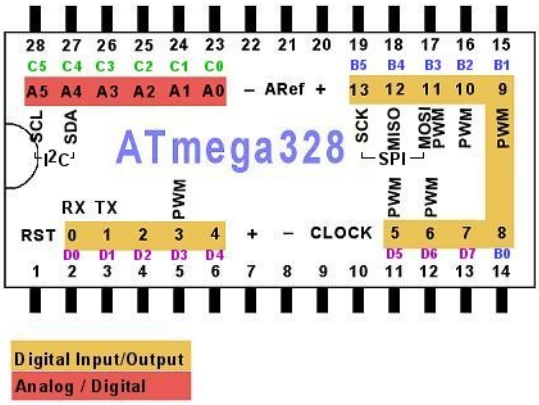

FIGURE 2 – lien entre les ports de la carte arduino et les bornes de la carte d'interfaces

2.1 Version 1

Lors de l'appuie sur le bouton poussoir, il faut que les leds soient allumées. Lors de la relâche, il faut que les leds s'éteignent.

2.2 Version 2

Dans la deuxième version, le comportement de la led branchée sur `D11` doit être le même que précédemment. Par contre la led branchée sur `D8` doit s'allumer lors de l'appuie sur le bouton poussoir et s'éteindre uniquement lors de l'appuie suivant sur le bouton poussoir.

2.3 Version 3

Maintenant, la led branchée sur `D8` doit clignoter lors d'un appuie sur le bouton poussoir et doit rester éteinte lors d'un deuxième appuie sur le bouton. Ralentissez la vitesse de clignotement et testez plusieurs fois votre programme. Quel est le problème ? Comment à votre avis le résoudre ?

3 Programmation avec interruption

Première chose, vous devrez inclure le fichier `<avr/interrupt.h>` à votre programme. Ensuite, souvenez-vous que l'utilisation des interruptions dans votre programme varie selon que vous utilisez un compilateur ou un autre... Pour les utilisateurs de `avr-gcc`, un *handler* se déclare comme ceci :

```
ISR(nom_interruption_vect)     // Interrupt handler for nom_interruption
{
        //do some stuff
}
```

Si vous utilisiez un autre compilateur comme `iccavr`, un *handler* se déclare comme une fonction classique mais il faudra ajouter une information au compilateur sous forme de `pragma` :

```
#pragma nom_du_handler ISR:numero_de_interruption
```

3.1 Utilisation d'une interruption sur changement d'état d'une broche

On veut le même fonctionnement que la version 3 faite précédemment. Cependant cette fois l'appuie sur le bouton déclenchera une interruption dont le handler se chargera de changer une variable globale.

On utilise l'interruption PCINT0. Positionner le bouton poussoir en conséquence. Il s'agit ensuite de paramétrer les bons registres du contrôleur afin d'autoriser l'interruption PCINT0.
Remarque : dans le handler de l'interruption, on commence en principe par désactiver l'interruption qui nous a déclenché. Ensuite pour des raisons de rebonds électrique lors de l'appuie sur le bouton poussoir, vous mettrez un petit délai dans le handler (normalement on élimine ces rebonds électroniquement).

3.2 Utilisation d'un timer avec interruption

Vous allez modifier le programme de la question précédente pour le faire maintenant fonctionner avec une interruption initiéeS par le Timer 16bit numéro 1(pages 115 à 145 des datasheets et plus particulièrement de 136 à 142). Pour ceci sachez qu'il vous est possible de générer une interruption lorsqu'un timer atteint une valeur prédéfinie (le vecteur d'interruption associé est nommé `TIMER0_COMPA_vect`). Cette valeur doit alors être mise dans un(des) registre(s) spécifiques (`OCR1A` dans notre cas). Pour calculer cette valeur de comparaison pour le timer 1, vous devez d'abord choisir la vitesse à laquelle le timer va évoluer (sa période). Pour cela un registre spécifique permet de choisir si la période du timer est celle de l'oscillator du circuit (qui est de 1/16MHz dans notre cas) ou à un multiple de cette période (table 16-5 des datasheets). Une fois la période du timer choisie, on sait que pour une division de la vitesse de 1024 le timer s'incrémentera tous les :

$$(1024/16MHz) = 64\mu s$$

Ainsi si vous comparez votre timer à une valeur de 15625 (soit 0x3D09 en hexadecimal), vous voulez attendre 1 seconde.
Dans votre programme vous configurerez les registres de sorte que la led branchée sur la borne `D8` change d'état toutes les secondes.

4 Un autre programme (nettement plus complet)

4.1 Version 1

Le but du prochain programme est d'afficher sur un ecran LCD la valeur de la luminosité ambiante. Pour ce faire on utilise un capteur de luminosité. Ce capteur renvoie une tension valeur analogique qui augmente en fonction de la luminosité. Comme on est informaticien on aime pas beaucoup les tensions continues. Il est donc nécessaire de convertir cette valeur en numérique. Nous utiliserons donc le convertisseur analogique numérique présent dans le micro controlleur (pages 252 à 268 des datasheets). Vous enleverez donc le bouton poussoir de la borne `A1` afin d'éviter tout problème. Vous brancherez le capteur de luminosité sur la borne `A5`. Il vous sera nécessaire de paramètrer le convertisseur. voici les paramètres à lui appliquer :

- tension de référence égale à $AV_{cc} \rightarrow$ `REFS0`=1 et `REFS1`=0
- utilisation de `ADC1`
- fréquence de conversion à 125KHz (*prescaler* à 128 dans le registre `ADCSRA`)
- trigger automatique, *free running mode* (registre `ADCSRA` et `ADCSRB`)

Une fois paramètré, vous pourrez selectionné et demarrer le convertisseur (registre `ADCSRA`). La valeur peut alors être lue dans `ADCW`.
Afin d'afficher cette valeur, nous allons utiliser l'écran LCD. Il vous faut le relier via le une nappe, au bus numéro 2. Vous êtes obligé d'utiliser ce bus car c'est celui utilisé dans le code suivant qui utilise une bibliothèque fournissant un object C++ facilitant l'utilisation de l'écran (sinon c'est difficile à utiliser !). Il vous faut donc ajouter ceci dans le code : `#include <LiquidCrystal.h>`. Voici une utilisation simple du LCD :

```
LiquidCrystal lcd(10, 11, 12, 13, 14, 15, 16); //init and link to the bus pins
lcd.begin(16, 2);  // configures it

lcd.clear(); //clears the LCD and positions the cursor in the upper-left corner
```

```
 lcd.setCursor(2,1);  // set to the 3th column and 2nd row
 lcd.print("I print !");
```

Pour une raison obscure, vous devrez aussi ajouter le code suivant :

```
extern "C" void __cxa_pure_virtual(void)
{
// call to a pure virtual function happened ... wow, should never happen ... stop
while(1)
      PORTB=0x00;
}
```

4.2 Version 2

On veut ajouter au comportement précedent une détection des chocs. Pour se faire on veut utiliser `D9` (i.e. `PB1`) comme source d'interruption exterieure (en utilisant de fait *PCINT2*). En effet il est possible de déclencher une interruption dont le vecteur d'interruption est `PCINT0_vect` lorsqu'un changement d'état sur cette patte est détecté. Nous bracherons donc le capteur de tilt sur cette borne. Notez que dans un premier temps je vous conseil d'utiliser le bouton poussoir pour plus de simplicité à simuler les chocs.
Lors de la détection d'un choc on veut afficher un message sur la ligne inférieur de l'écran LCD. Ce message devra s'afficher pendant quelques secondes, durant lesquelles l'affichage de la luminosité devra continuer à être mis à jour.