

Systèmes embarqués

Programmation micro-contrôleur

Sans OS

Malgré la distance je vous demande de travailler en binôme, un sur le matériel et l'autre sur le simulateur. Cherchez ensemble à comprendre ce qu'il se passe et comment résoudre les problèmes que vous rencontrez.
Comme nous allons faire le tout à distance, j'ai mis à côté de certaines sections un numéro (par exemple "Installation de la suite logicielle" a le numéro #1). Ceci réfère au sondage sur slack pour le niveau d'avancement de chacun. Ainsi, je vous remercie de bien vouloir mettre à jour la question que vous êtes en train de traiter dans le sondage slack.

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. Mais attention, ne vous amusez pas à les lire en entier sinon vous n'aurez jamais le temps de faire le moindre programme dans les prochaines semaines. 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 modifier pour atteindre les objectifs donnés dans la suite.

Un résumé des fonctions des pattes du micro contrôleur est donné figure 1. Par exemple pour la patte 2 du micro-contrôleur, PD0, cela réfère au bit 0 du port D. Cette patte partage plusieurs fonctionnalités dont PCINT16 (Pin Change Interrupt #16, permettant de réagir à un changement d'état de la patte, voir section 13 des datasheets) et RXD (réception des données séries de L'USART : *Universal Synchronous and Asynchronous serial Receiver and Transmitter* dont la description est faite en section 20 des datasheets).

1 Premiers pas

1.1 Installation de la suite logicielle (poll #1)

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

- compilateur
 - **avr-gcc** : sous linux, les paquets existent pour la majorité des distributions (sous le nom gcc-avr). Utilisez les manageurs de paquets de votre distribution en premier lieu (apt-get, yum, etc sont vos amis. Par exemple apt-cache search gcc-avr cherche les paquets contenant gcc-avr dans leur nom).
 - **avr-libc** : similaire au point précédent.
- *linker/uploader*

(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)

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

— avrdude : devrait aussi être dans les paquets des distributions linux les plus courantes.

1.2 Installation matérielle ou simulateur (poll #2)

Selon le cas vous avez le matériel Arduino que l'on vous a distribué ou pas. Les deux sections ci dessous traitent chacun de ces cas. À noter que normalement vous pouvez travailler sur le même programme avec votre binôme et que les deux doivent fonctionner de la même manière (à quelques tous petits changements prêts pour le LCD).

1.2.1 carte Arduino

Durant ces exercices, vous allez utiliser la carte Arduino et une carte permettant de relier divers capteurs/actionneurs facilement (appelée *Shield*). L'ensemble est alimenté par un câble USB. C'est également ce câble qui permettra de transférer votre programme sur le micro contrôleur et de communiquer avec en série.

Pour commencer, prenez 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). Faites également attention à ne pas laisser les capteurs/actionneurs faire des court-circuits ; entre eux ou avec la carte.

Installation (à faire sans que le câble USB soit branché) : vous devez mettre la carte *Shield* sur le micro-contrôleur. Normalement c'est fait pour ne pas pouvoir se tromper. Cependant faites bien attention à ne pas tordre les pattes et vous assurez qu'elles sont bien toutes en face de leur logement avant de les insérer. Une fois ceci fait, vous voyez que vous avez, sur le dessus du shield, accès aux pattes D2 à D8 du port D. Ce sont des pattes bi directionnelles numériques (digital), i.e., pouvant servir d'entrées/sorties numériques. Vous avez aussi accès aux pattes C0 à C3 du port C (nommée A0 à A3) qui sont des pattes pouvant servir d'entrées/sorties analogiques. Vous avez aussi accès aux broches permettant la communication sur un bus I2C. C'est un bus de terrain, permettant une communication série entre un maître et différents esclaves (<https://fr.wikipedia.org/wiki/I2C>). L'afficheur LCD que nous utiliserons plus tard est un esclave pouvant être contrôlé sur le bus I2C. Remarquez que vous avez aussi accès à l'USART (nommé UART ici) correspondant aux pattes D0 et D1. Nous n'utiliserons pas l'USART au travers de cette fiche mais au travers de la connexion USB servant par ailleurs à uploader le programme dans le micro-contrôleur.

Dans un premier temps nous allons utiliser 2 leds et un bouton poussoir. Attention, dans le kit fourni, les leds ne sont pas montées sur la petite carte associée. Comme une led est une diode électro-luminescente, alors elle ne laisse passer le courant que dans un seule sens. Pour la monter correctement, remarquez qu'elle n'est pas ronde mais qu'elle possède un côté plat à sa base ; tout comme le dessin sur la carte associée.

Pour les premières quesdtions du TP, nous allons brancher la led rouge sur D2, la led verte sur D3 et un bouton poussoir sur D4. Plus tard, nous brancherons aussi l'afficheur LCD sur un des ports I2C.

Vous pouvez ensuite brancher votre arduino à votre PC via le câble USB. Il ne va rien se produire car l'arduino n'est pas programmé. Vous pouvez donc passer au HelloWorld section 1.3.

1.2.2 simulateur SimulIDE

Pour simuler les composants électroniques dont la carte Arduino, vous allez utiliser *SimulIDE* : <https://simulide.blogspot.com/>. Commencez par télécharger la version qui correspond à votre distribution. Ensuite lancer là et ouvre le fichier *TP1.Simu* se trouvant sur ma page web. Vous devriez avoir le montage ci dessous :

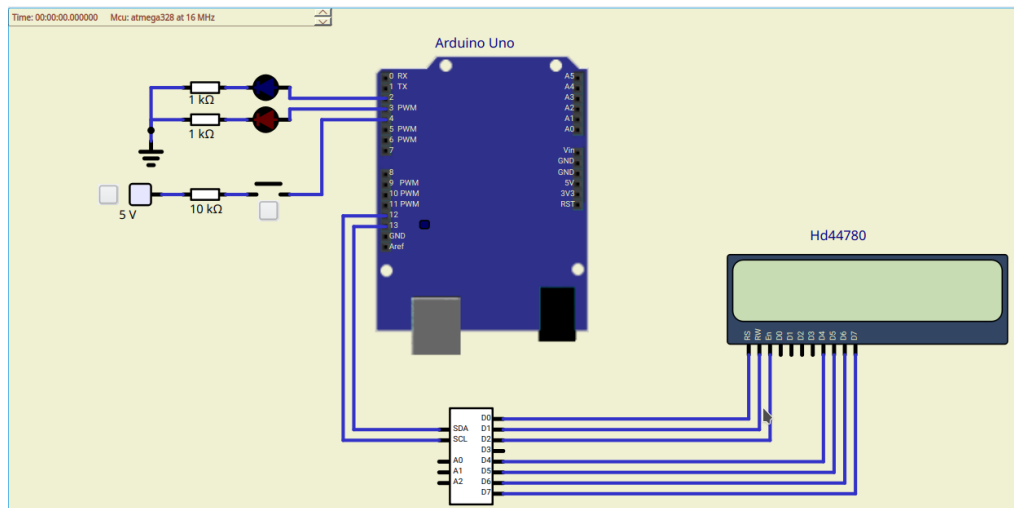



FIGURE 2 – Vue du cablage relative aux premières questions du TP1 dans SimulIDE

Pour démarrer la simulation, il suffit de mettre le montage sous tension en cliquant le bouton . Cela ne va, dans un premier temps rien faire car l'arduino n'est pas programmé. Pour le programmer, il faudra faire un clic droit dessous et choisir : “Load firmware”. Il faudra alors naviguer vers le “.hex” qui résultera de la compilation.

1.3 Hello World sur la carte arduino (poll #3)

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 œuvre pour l'utiliser. De ce fait, l'équivalent du traditionnel “Hello World” est “LED Blink”. Ce dernier, comme vous l'avez tous compris consiste à faire clignoter une diode électro-luminescente (il est simple de brancher et de piloter une led à branchée à une sortie numérique du micro-contrôleur.). C'est donc le programme qui vous est fournis pour vos premiers tests. Il fait clignoter les deux leds précédemment branchées sur PD2 et PD3. Téléchargez le sur ma page (ou regardez dans le zip) ainsi que le Makefile associé. Avant de l'examiner, essayez le afin de vous assurer que la suite d'outils est correctement installée.

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 .cpp 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 comme `/dev/ttyACM0` par votre OS. Si ce n'est pas le cas, ajustez...

charger dans le simulateur : Allez chercher le fichier .hex dans le dossier Build depuis le simulateur (click droit sur l'arduino puis "Load firmware").

Vérifiez que la les deux leds clignotent...

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 ?
- Qu'est-ce que DDRD ? Que se passe t'il si je mets une led en D5 ? comment ajuster le programme correspondant ?
- Que se passe t'il si l'on ajoute dans le Makefile l'option '-Os' (c'est la lettre O pas le chiffre, à mettre au début de la ligne des "CPPFLAGS") qui permet de mettre en œuvre les optimisations de gcc ? Testez (en faisant des "make clean" entre chaque changement dans le Makefile) et expliquez (indice : regardez la taille du code avec et sans l'optimisation)
Modifier votre programme pour qu'il fonctionne correctement malgré l'option d'optimisation. Remarquez le gain en taille du code.
- quelle est la manipulation minimum du registre DDRD (ligne 17) afin que celui ci corresponde à notre setup avec une led en D2, une en D3 et un bouton en D4 ?
- regardez les bonnes pratiques...
- ... ?

Une fois que vous avez vraiment compris ce programme, passez à la suite.

2 Premier (deuxième ?) programme (sans interruptions)

2.1 Version 1 (poll #4)

Créez un programme qui se comporte comme suit. 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.

Pour ce faire, pensez à lire la section 14.1 des datasheets en entier ; en particulier le troisième paragraphe. Vous pourrez aussi lire la section 14.2.4.

2.2 Version 2 (poll #4 aussi)

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

2.3 Version 3 (poll #5)

Maintenant, la led rouge doit se mettre à clignoter lors d'un premier appui sur le bouton poussoir et doit rester éteinte lors d'un deuxième appui 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 (poll #6)

On veut le même fonctionnement que la version 3 faite précédemment. Cependant cette fois l’appui sur le bouton déclenchera une interruption dont le *handler* se chargera de changer une variable globale¹. **Lisez l’introduction du chapitre 13 des datasheets.** On utilise l’interruption PCINT2 (Pin Change Interrupt 2) déclenché par un changement d’état de PCINT20. Positionner le bouton poussoir en conséquence. Il s’agit ensuite de paramétrer les bons registres du contrôleur afin d’autoriser l’interruption PCINT2. Ceux-ci sont décrits dans le chapitre 13 des datasheets.

Remarque : dans le handler de l’interruption, on respecte habituellement les règles suivantes :

- on reste le minimum de temps possible !
- Souvent, on commence par désactiver l’interruption qui nous a déclenché, on fait le traitement, puis on réactive l’interruption. Le “souvent” signifie lorsque que l’on ne veut pas traiter des rafales d’interruptions TRÈS rapprochées dans le temps. Voir : “cela dépend de la sémantique de l’interruption” dans le cours.
- pour des raisons de rebonds électrique lors de l’appui sur le bouton poussoir, vous mettez un petit délai dans le handler; cette fonctionnalité est habituellement nommée *debounce*. Remarquez que ce délai est en contradiction avec le premier point. Donc lorsque c’est possible on préfère éliminer ces rebonds électroniquement (<http://www.labbookpages.co.uk/electronics/debounce.html>).

3.2 Utilisation d’un timer avec interruption (poll #7)

Vous allez modifier le programme de la question précédente pour le faire maintenant fonctionner avec une interruption initiée 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 `TIMER1_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’oscillateur 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. Attention, par défaut le timer ne revient pas à zéro une fois qu’il a atteint la valeur de `OCR1A`; il faut donc être sur de se mettre dans le mode CTC (Clear Timer on Compare match) basé sur `OCR1A`.

Dans votre programme vous configurerez les registres de sorte que la led rouge change d’état toutes les secondes. Vous ajouterez aussi le code pour que la led verte change d’état à chaque appui sur le bouton. Remarquez que cette fois la led rouge est bien périodique et indépendante du délai de debounce et du nombre de fois où vous appuyez sur le bouton.

1. vous avez lu variable globale et vos cheveux se sont hérissés ? Il n’y a pas de quoi. C’est la manière classique de communiquer entre handler et code principale en programmation micro-contrôleur. Cependant vous avez eu raison de réagir car une personne attentive ira lire cette document claire et utile : https://www.nongnu.org/avr-libc/user-manual/group__util__atomic.html

4 Utilisation de bibliothèques Arduino (poll #8)

La plateforme Arduino a l'intérêt de posséder beaucoup de petits matériels compatible avec la plateforme et pour laquelle des informaticiens ont fait des bibliothèques permettant de cacher la complexité de configuration des différents registre du micro contrôleur. C'est les cas par exemple pour utiliser le module LCD *Grove* qui vous a été fourni ou le LCD du simulateur *simulIDE*. Les deux sont des modules esclaves I2C. J'ai légèrement modifié l'API du LCD du simulateur afin qu'il corresponde à celui du module Grove. À l'exception de la bibliothèque importée, de la déclaration du LCD (et de la compilation de la bibliothèque bien sûr), vous pouvez utiliser le même code sur l'un et sur l'autre.

4.1 LCD

Afin de tester ceci, récupérer le code d'exemple du LCD sur ma page web. Voyez qu'il comprend un répertoire *arduinoLibsAndCore*. C'est une extraction que je vous fourni afin de pouvoir utiliser les bibliothèques Arduino. Normalement la librairie est déjà compilée et se nomme *libres.a*. Si un problème apparaît, vous pouvez la recompiler en faisant un *make clean; make libarduino*.

Dans l'exemple fourni la librairie est ajoutée à la compilation dans le Makefile. Cela permet d'utiliser la librairie du LCD. Vérifiez que tout marche correctement.

4.2 Serial

Afin de déboguer plus facilement (entre autre), il est utile de communiquer entre le code Arduino et le PC. Pour ce faire on peut utiliser la liaison Série. Une bibliothèque bien utile d'Arduino est `<Serial.h>`. Elle permet d'établir facilement une connexion bi-directionnelle.

Encore une fois, télécharger le code correspondant et testez le. Remarquez que pour faciliter la lecture des données sur le port série du PC ainsi que l'injection de données, j'ai développé un petit utilitaire en python. Vous pouvez le lancer en faisant un *make monitorSerial*. Vous pourrez alors visualiser les données envoyées par l'arduino et lui en envoyer également (la touche ESC permet de quitter l'outil). Si votre Arduino n'est pas sur le port `/dev/ttyACM0` alors changer la config au début du script python. Pour ceux qui sont dans le simulateur, il existe l'outil équivalent. Pour l'ouvrir, click droit sur l'Arduino et "open Serial Monitor".

5 Un vrai programme (nettement plus complet)

Maintenant que vous avez vu les base de l'utilisation d'un micro contrôleur sans OS avec la configuration des registres, les handlers d'interruption, la notion de timer et l'utilisation de bibliothèques Arduino ; vous êtes prêt pour un petit programme.

5.1 Version 1 (poll #9)

Le but du prochain programme est d'afficher sur un écran LCD la valeur de la luminosité ambiante. Pour ce faire on utilise un capteur de luminosité. Ce capteur renvoie une valeur analogique (une tension) 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 contrôleur (pages 252 à 268 des datasheets). Vous brancherez le capteur de luminosité sur la borne A0. Il vous sera nécessaire de paramétrer le convertisseur. voici les paramètres à lui appliquer :

- tension de référence égale à AV_{cc} → `REFS0=1` et `REFS1=0`
- utilisation de `ADC0`
- fréquence de conversion à 125KHz (*prescaler* à 128 dans le registre `ADCSRA`)
- trigger automatique, *free running mode* (registre `ADCSRA` et `ADCSRB`)

Une fois paramétré, vous pourrez sélectionner et démarrer le convertisseur (registre `ADCSRA`). La valeur peut alors être lue dans `ADC`.

Afin d'afficher cette valeur, nous allons utiliser l'écran LCD. La mise à jour de la valeur lue se fera tous les 1200ms. De plus, la led rouge changera d'état à chaque mise à jour.

Le capteur de luminosité est fourni dans le kit Grove. Cependant, il peut être difficile de faire varier la luminosité ambiante correctement lors des tests. De plus, pour ceux qui utilisent le simulateur, il n'y a pas de capteur de luminosité. Ainsi, que ce soit en simulation ou en réel, vous pouvez utiliser un potentiomètre ; i.e., une résistance variable permettant lorsqu'elle est alimentée de faire varier la tension à ses bornes lorsque l'on tourne son bouton. Vous pourrez ainsi "mock" le capteur de luminosité.

5.2 Version 2 (poll #10)

On veut ajouter au comportement précédent la possibilité de savoir depuis combien de temps le montage est sous tension. Pour se faire on veut utiliser un bouton poussoir comme source d'interruption extérieure lorsqu'un changement d'état sur la patte correspondante est détecté.

Lors de l'appui sur le bouton on veut afficher le "uptime" de manière lisible (heure :minutes :secondes :ms) sur la ligne inférieure de l'écran LCD. Ce message devra s'afficher pendant 4 secondes, durant lesquelles l'affichage de la luminosité devra continuer à être mis à jour. De plus, la led verte restera allumée pendant ces 4 secondes et s'éteindra en même temps que le "uptime" s'enlèvera de l'écran.

Attention, n'essayez pas d'atteindre directement l'objectif. Faites plusieurs petits tests pour comprendre ce qui marche et pas.