TP n°5

Informatique embarquée

Vincent Ruello

À rendre avant mercredi 19/11/2025 – 18:00 (CET)

Le but de ce TP est de découvrir la manipulation de signaux analogiques ou pseudo-analogiques, en entrée comme en sortie.

Contexte

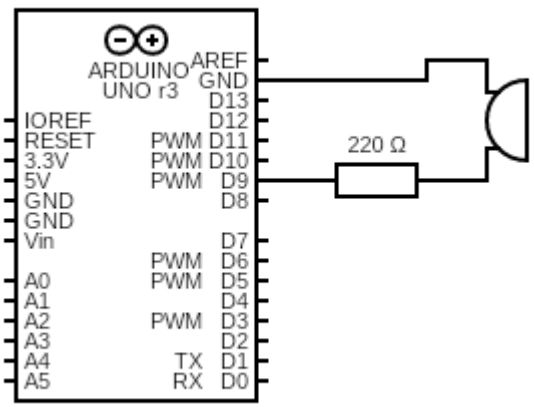
La carte Arduino UNO est une plateforme open-source de développement permettant de programmer un ATmega328P. La documentation du microcontrôleur ATmega328P est disponible ici[1]. AVR Libc est une bibliothèque open-source dont le but est de fournir une libc utilisable sur l'architecture AVR avec GCC. La documentation est disponible ici[2].

Partie 1. Utilisation d'un buzzer passif

Le but de cette partie est d'utiliser un buzzer passif pour émettre des « sons », voir même jouer une « mélodie ».

Un buzzer passif se comporte comme un haut-parleur : un signal périodique en entrée fait vibrer une membrane, qui émet ainsi une onde sonore. La fréquence de l'onde sonore émise est identique à la fréquence du signal donné en entrée.

Le buzzer passif peut être relié au microcontrôleur en série avec une résistance de 220 ohms selon le schéma suivant :



1 http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf

2 https://www.nongnu.org/avr-libc/user-manual/index.html

On souhaite générer un signal carré à l'aide du timer Counter1.

1. Identifier la broche de sortie OC1A sur le microcontrôleur (lettre et numéro), puis son numéro sur la carte Arduino R3.
2. Dans quel mode peut-on utiliser le timer Counter1 pour générer un signal carré sur sa sortie OC1A ?
3. Implémenter dans un fichier séparé (par exemple `buzzer.c`) les fonctions :
   - `buzzer__init()` : initialise le contrôle du buzzer
   - `buzzer_play(int16_t freq)` : envoie un signal carré de fréquence `freq` au buzzer. Le signal est envoyé en continu jusqu'à l'appel à la fonction `buzzer__stop()`.
   - `buzzer__stop()` : arrête l'envoi du signal (démarré par `buzzer__play`).
4. Reproduire le montage présenté dans le schéma précédent. **Faire valider votre montage par l'encadrant avant de le brancher au microcontrôleur.**
5. Écrire un programme qui, à l'aide des fonctions précédentes, fait jouer au buzzer en boucle le motif suivant : note A5 (de fréquence 880 Hz) pendant 1 seconde puis silence pendant 1 seconde.

<u>Pour aller plus loin :</u>

6. Écrire un programme qui, à l'aide des fonctions précédentes, fait jouer au buzzer en boucle le motif suivant :

   ```
   E5, E5, F5, G5, G5, F5, E5, D5, C5, C5, D5, E5, E5, SILENCE, D5, D5,
   SILENCE, E5, E5 , F5, G5, G5, F5, E5, D5, C5, C5, D5, E5, D5, SILENCE,
   C5, C5, SILENCE, SILENCE
   ```

   Avec :

   - E5 : 659 Hz pendant 300ms, puis 100ms de silence.
   - F5 : 698 Hz pendant 300ms, puis 100ms de silence.
   - G5 : 784 Hz pendant 300ms, puis 100ms de silence.
   - D5 : 587 Hz pendant 300ms, puis 100ms de silence.
   - C5 : 523 Hz pendant 300ms, puis 100ms de silence.
   - SILENCE : 400ms de silence

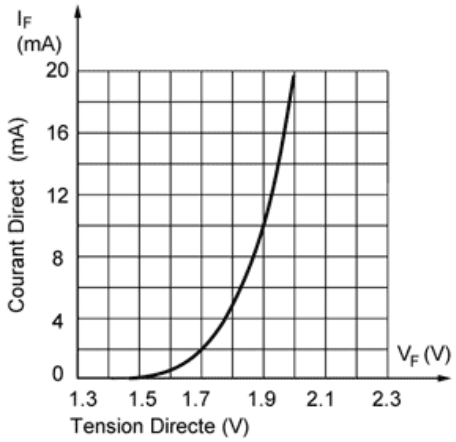
Partie 2. Utilisation d'une LED externe

Le but de cette partie est de réaliser un circuit électrique afin de piloter l'état d'une LED 5mm avec le microcontrôleur.

Une LED (Light-Emitting Diode) est avant tout une diode : un dipole polarisé qui ne laisse passer le courant que dans un sens, de son anode vers sa cathode (notée K). On peut parfois différencier les deux broches par leur taille : l'anode est la plus grande, la cathode est la plus petite.

L'intensité lumineuse produite par une LED dépend de l'intensité électrique qui la traverse. En général, les LEDs 5mm fournissent une intensité lumineuse nominale avec une intensité de 20mA.

La figure suivante représente la caractéristique typique d'une LED :

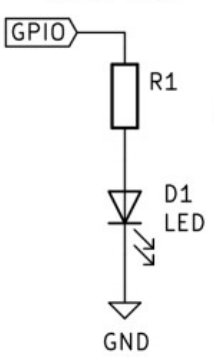


Chaque couleur de LED a une courbe caractéristique légèrement différente due à sa composition qui est précisée dans sa *datasheet*. Les points de fonctionnement associés à une intensité de 20mA sont listés dans le tableau suivant :

5mm LEDs		
Color	Forward Voltage	Forward Current
White	3.1V	20ma
Red	2.1V	20ma
Blue	3.4V	20ma
Green	3.4V	20ma
Yellow	2.1V	20ma

La carte Arduino alimente le microcontrôleur avec une tension 5V. Les broches de sortie, lorsqu'elles sont placées à l'état haut, sont au potentiel électrique 5V (et au potentiel 0V lorsqu'elles sont à l'état bas).

Afin de se placer sur le point de fonctionnement de la LED associé à l'intensité 20mA, il est nécessaire de réduire la tension aux bornes de la LED par rapport à celle fournie entre les broches de sortie du microcontrôleur et la broche GND. Pour cela, on branche une résistance en série avec la LED, comme montré dans le schéma suivant :



On peut calculer une valeur de résistance « idéale » en fonction de la couleur de LED. On utilisera ici une résistance d'environ 150 Ohms.

1. Connecter votre circuit à la broche de votre choix (par exemple PD4) puis adapter un des programmes du TP 3 pour faire clignoter la LED.

Faire valider votre montage par l'encadrant avant de le relier au microcontrôleur.

Partie 3. Clignotement continu

Dans les TPs précédents, la LED intégrée s'allumait puis se coupait. On veut désormais faire varier l'intensité lumineuse de façon continue. Pour cela, on va utiliser la technique de la modulation de largeur d'impulsion (Pulse Width Modulation ou PWM en anglais) pour synthétiser un signal pseudo-analogique.

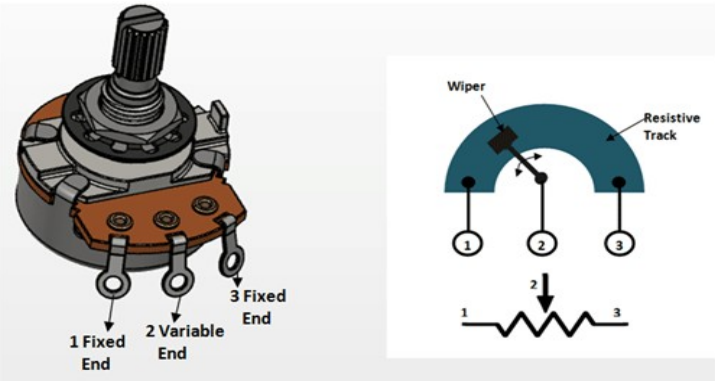
1. Faire un schéma d'une modulation de largeur d'impulsion d'un signal avec un rapport cyclique (*duty cycle*) de 0, 0.25, 0.50, 0.75 puis 1.
2. On utilisera le timer Counter0 pour générer notre sortie en utilisant le mode Fast PWM et le comparateur A (sans inversion). Avec quelle valeur faut-il configurer OCR0A pour réaliser les *duty cycles* 0, 0.25, 0.50, 0.75 et 1 ?
3. Identifier la broche à laquelle est connectée la sortie du module comparateur A du timer Counter0. Connecter le circuit de la partie 1 à cette broche.
4. Ecrire un programme qui allume la LED externe avec 10 % d'intensité lumineuse.
5. Il est possible de modifier OCR0A lors de l'interruption TOV0. Écrire un programme qui fait clignoter avec une intensité continue la LED intégrée. On choisira un prescaler approprié pour pouvoir observer le clignotement. Pour rappel, la fréquence du signal émit est égale à $F_{clk\_I/O}$ / (N*256) où N représente le prescaler utilisé (1, 8, 64, 256 ou 1024).

Pour aller plus loin :

6. Améliorer votre programme pour fixer la fréquence de clignotement à 1 seconde.

Partie 4. Mesure d'une entrée analogique avec l'ADC

Dans certains cas, on souhaite mesurer la valeur d'une entrée analogique. Un exemple est l'utilisation d'un potentiomètre.
De façon macroscopique, un potentiomètre se comporte comme une résistance variable. Il est composé de trois bornes dont une est reliée à un curseur se déplaçant sur une piste résistante terminée par les deux autres bornes. Les potentiomètres utilisés en TP correspondent au schéma suivant :



Le potentiel électrique de la broche 2 est compris entre ceux des proches 1 et 3 et varie selon la position du curseur.

Le périphérique ADC (Analog-to-Digital Converter) permet d'échantilloner la valeur du potentiel électrique imposé sur une broche sur 10 bits. Dans le cadre de ce TP, on se contentera d'une résolution de 8 bits.

1. Écrire un programme qui fait varier l'intensité de la LED intégrée en fonction de la position du curseur du potentiomètre.

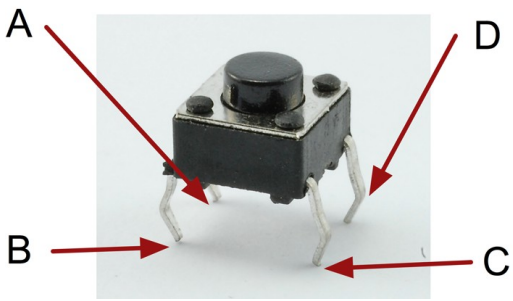
Pour aller plus loin :

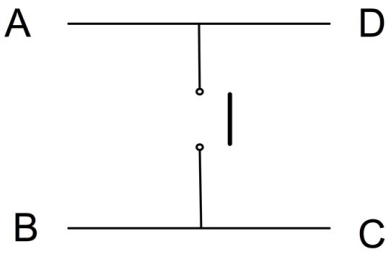
2. Écrire un programme qui fait varier la fréquence de clignotement de la LED intégrée en fonction de la position du curseur du potentiomètre.

Partie 5. Debouncing (bonus)

Le but de cette partie est de contrôler l'état d'activation de la LED à l'aide d'un bouton poussoir : lorsque le bouton est pressé, l'état de la LED doit changer (allumée / éteinte).

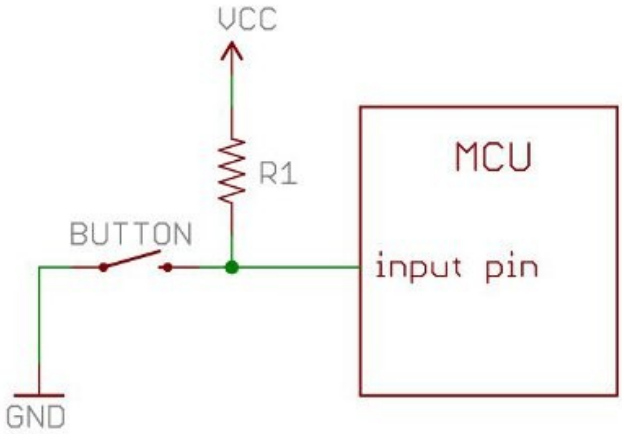
Un bouton poussoir se présente sous la forme d'un interrupteur. L'interrupteur est à l'état ouvert « au repos », et est fermé lorsque le bouton est pressé.





Pour rappel, une GPIO peut être mise en mode « Entrée ». Le programme peut alors lire l'état logique présent sur la broche.

On propose d'utiliser le montage suivant :



La ligne est maintenue à l'état logique haut à l'aide d'une résistance R1 appelée résistance de Pull-Up . Une telle résistance est inclue dans chaque GPIO de l'ATmega328P et peut être configurée à l'aide du registre IO PORTxn. Le bouton connecte la ligne à la masse.

Au repos, l'interrupteur est ouvert. La broche d'entrée du microcontrôleur est donc reliée à VCC par la résistance de pull-up et est au potentiel VCC. Lorsque le bouton est pressé, l'interrupteur se ferme : la broche d'entrée est alors mise au potentiel de la masse (0). On détecte ainsi l'appui sur le bouton en étudiant le changement de l'état logique présent sur la GPIO.

Pour éviter tout dommage en cas de mauvaise configuration de la GPIO (sortie à la place d'entrée), on ajoutera à ce montage une résistance R2 d'environ 500 ohms en série sur la ligne qui relie la broche d'entrée et la masse.

1. Écrire un programme « simple » qui, dans une boucle infinie, lit l'état de la broche en entrée. Si l'état lu est l'état bas (le bouton est pressé), l'état de la LED doit être changé. On veillera à **configurer la GPIO utilisée en entrée** et à activer sa résistance de Pull-Up interne.
   On réalisera le montage décrit précédemment à l'aide du bouton et d'une résistance de sécurité de 470 Ohms. **Faire valider le montage avant de le connecter au microcontrôleur.**

Appuyer à plusieurs reprises sur le bouton. Normalement, l'état de la LED devrait changer de façon assez erratique. Selon vous, quelle peut en être la cause ?

Ce phénomène est appelé *bouncing*. Il existe plusieurs techniques permettant d'y répondre : certaines sont matérielles, d'autres sont logicielles.

Une méthode logicielle permettant de répondre à ce problème consiste à s'assurer qu'un état reste stable pendant un certain temps avant de le prendre en compte. On propose d'implémenter l'algorithme suivant sous la forme d'une fonction `debounce` :

```
read current button state
if (current button state != button state)
      Increase counter
      if (counter >= 4)
            change button state
            reset counter
            if (button state is low)
                  raise button pushed flag
            end
      end
else
      reset counter
end
```

Les variables `button_state`, `counter`, et `button_pushed_flag` sont des variables globales.

La fonction `debounce` doit être appelée régulièrement, par exemple toutes les 15ms. Pour cela, on propose d'utiliser le Watch Dog Timer. La boucle principale vérifiera à chaque itération l'état du flag button_pushed, et le cas échéant fera basculer l'état de la LED.

2. Implémenter le mécanisme décrit précédemment.