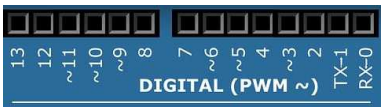
III - Gestion des entrées et sorties avec Arduino

a) Configuration (pinMode)



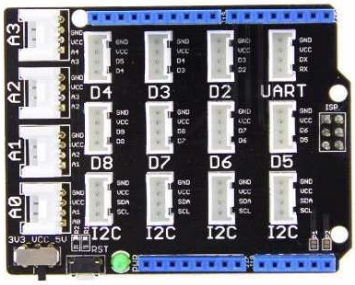
Dans l'Arduino UNO on trouve des ports numérotés :

- de 0 à 13 pour les ports digitaux classiques : ils peuvent servir de ports d'entrée ou de sortie ;
- de A0 à A5 pour les ports d'entrée analogiques.
  Ils peuvent mesurer des tensions comprises entre 0 et 5 volts (avec une précision de 10 bits soit 1024 valeurs possibles – entre 0 et 1023).

Les signaux véhiculés par les ports 0 à 13 ne peuvent prendre que deux états HAUT (5 Volts) ou BAS (0 Volt) (courant de 40 mA maximum par sortie). HAUT peut être traduit par HIGH ou 1 et BAS par LOW ou 0.
Les ports 0 et 1 sont réservés pour la liaison USB et ne sont donc pas utilisés (RX et TX sont utilisés pour gérer les flux de données entrants et sortants).
Les ports 3, 5, 6, 9, 10 et 11, repérés par un ∼, peuvent être utilisés en sortie PWM (Pulse Width Modulation), pour faire varier la luminosité d'une LED.

Les ports A0 à A5 étant toujours des entrées, aucun paramétrage supplémentaire ne sera demandé.
Par contre pour les autres ports, il faut préciser leur état. Cela se fait avec la fonction pinMode.
Par exemple :

- `pinMode(7,OUTPUT);` indiquera que le port 7 sera une sortie (par contre les deux seules valeurs qui pourront être prises sont HIGH et LOW car ce n'est pas une sortie PWM) ;
- `pinMode(2,INPUT_PULLUP);` indique que le port 2 sera une entrée. Par contre PULLUP est réservé pour un bouton poussoir (et même conseillé dans ce cas).



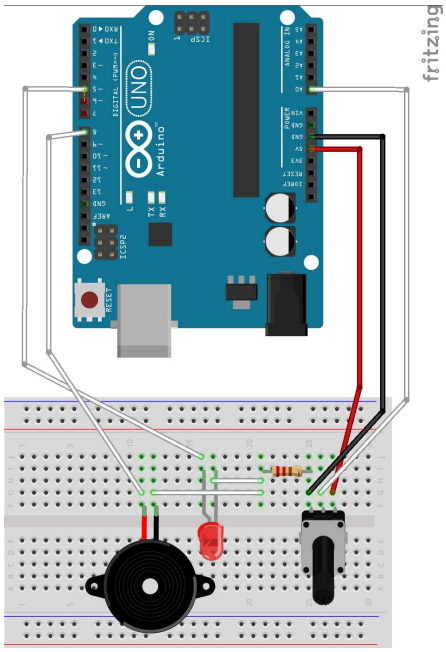
Remarque : En classe, les cartes UNO dont nous disposons se présentent avec une protection. Cela permet un usage plus sûr (évitant notamment de griller des composants), mais le nombre de ports est dans ce cas plus limité. Il n'y a que 7 ports digitaux nommés de D2 à D8 (leur numéro correspondant aux numéros 2 à 8, y compris pour les sorties PWM).
Par ailleurs, il n'y a que quatre ports d'entrée analogiques numérotés de A0 à A3.

b) Lecture et écriture

En Arduino, on utilise les fonctions `Read` et `Write` précédées du type de donnée (`analog` ou `digital`).

1. Exemple avec un potentiomètre, une LED et un buzzer.

   On construit un montage permettant d'allumer une LED ou de faire fonctionner un buzzer suivant la tension fournie au circuit. La tension initiale étant de 5 V, le potentiomètre permet de la faire varier de 0 à 5 V.
   Si elle est inférieure à 2,5, alors le buzzer fonctionne. Sinon, c'est la LED qui s'allume et la lumière émise est proportionnelle à la tension.
   Voici le montage et le code :



La résistance protège la LED (sans quoi, elle grillerait directement). On utilise ici une LED de 150 Ω (ou une valeur proche).

Avec le dispositif utilisé en classe, on n'aura pas tous ces soucis de branchements car la LED est déjà protégée.
De plus, les câbles ne sont pas uniques comme ceux de la figure. Les branchements de chaque composants se font directement sur les numéros des ports.

```
int laLED = 5;
int leBuzzer = 8;
int ledValue;
void setup () {
    // put your setup code here, to run once:
    Serial.begin (9600);
    pinMode (laLED, OUTPUT );
    pinMode (leBuzzer, OUTPUT );
}
void loop () {
    // put your main code here, to run repeatedly:
    // Mesure la tension sur la broche A0
    int valeur = analogRead (A0);
    int valMax = 683;
    // Transforme la mesure (nombre entier) en tension
    via un produit en croix
    float tension = valeur*5.0/valMax;
    if (tension>2.5){
       ledValue= map (valeur, valMax/2, valMax, 0, 255);
       analogWrite (laLED,ledValue);
       digitalWrite (leBuzzer, LOW );
    } else {
       digitalWrite (laLED, LOW );
       digitalWrite (leBuzzer, HIGH );
    }
    Serial.println (tension);
    Serial.println (valeur);
    Serial.println (ledValue);
    delay (250);
}
```

Trois variables sont utilisées de manière globale.

Dans la fonction `setup()`, la fonction `Serial.begin` sert juste à obtenir des informations sur l'ordinateur (cela sert avant tout à vérifier si les valeurs correspondent au comportement de la LED et du buzzer). La valeur 9600 sera toujours celle qu'il faudra utiliser.

Dans cette fonction, on déclare également les deux ports, celui de la LED et celui du buzzer, et le fait que ce sont des sorties (ces composants sont des actionneurs).

Le choix d'une sortie PWM pour la LED est important pour faire varier la luminosité en fonction de la tension obtenue.

Dans la fonction `loop()`, `analogRead` permet de récupérer la tension sur la broche A0. Cette tension n'est pas exprimée en volts mais sous la forme d'un nombre compris entre 0 et `valMax` qu'on peut déterminer en regardant en console les valeurs prises par la variable `valeur`.
La variable tension est juste une conversion de cette valeur en volt (par un simple produit en croix).

Dans ce programme, on utilise la fonction map qui est très pratique : elle convertit dans l'intervalle [0 ; 255] une valeur comprise ici dans un autre intervalle [`valMax/2` ; `valMax`].

L'intensité de la LED s'écrit au format numérique sur 8 bits, donc c'est bien une valeur entière entre 0 et 255. `ledValue` étant l'intensité de la lumière à transmettre à la LED, on utilise deux fonctions :

- `digitalWrite` qui renvoie l'état LOW ou HIGH à la LED (LOW pour l'éteindre) ;
- `analogWrite` qui donne au format analogique l'intensité de la lumière que la LED doit émettre quand elle est allumée.

À la fin, on affiche sur l'ordinateur la tension obtenue (grâce à `Serial.print`), ainsi que `valeur` et `ledValue` et on fait une pause de 250 ms (on reprend ensuite le début de la boucle).

Comment modifier ce programme (supprimer et changer des lignes) pour qu'on n'ait plus qu'un état allumé/éteint des LED suivant que la tension obtenue est inférieure ou supérieure à 2,5 V ?