**Un exemple de microcontrôleur : la carte Arduino**

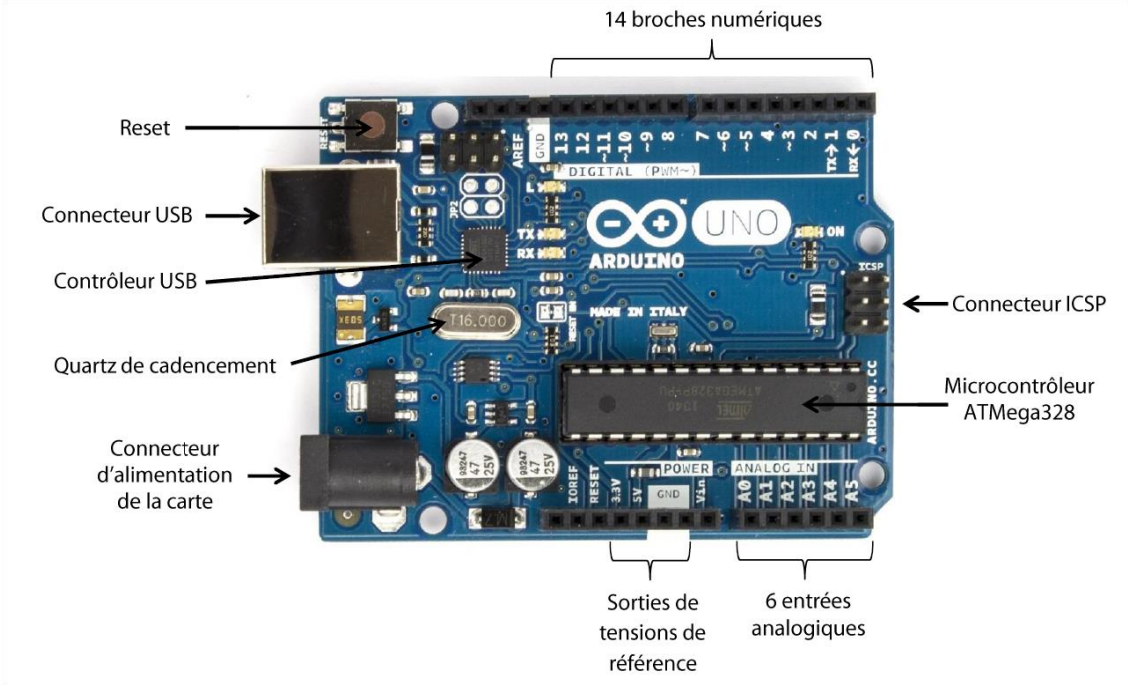
Il s'agit ici de se familiariser avec l'utilisation d'un microcontrôleur à travers quelques exemples simples d'utilisation.

## I Présentation (lecture)

Un microcontrôleur est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur, principalement le processeur, des mémoires et des interfaces d'entrées sorties. Ces dernières permettent aussi bien de récupérer des informations de capteurs (de température, pression, …) que de commander des systèmes (LED, moteur, …). Ils sont fréquemment utilisés dans les systèmes embarqués (automobiles, télécommandes, appareils de bureau, électroménager, jouets, téléphonie mobile, …) mais aussi au laboratoire comme la carte SYSAM SP5 utilisée en TP. La carte pourra en outre être avantageusement mise en œuvre pour les Tipe.

Parmi les microcontrôleurs très utilisés figure la carte Arduino qui était à l'origine un projet d'étudiants de l'école de design d'Ivrée en Italie. Elle emprunte son nom au « Bar di Re Arduino » (« bar du roi Arduin »), lieu de réunion des concepteurs de la carte.

Nous utiliserons ici le modèle de base, la carte « Arduino Uno ». L'architecture est détaillée sur la photographie ci-dessous :




Cette carte fonctionne, entre-autres, grâce à :
- un microcontrôleur ATMega328 ;
- un quartz 16MHz (horloge de cadencement) ;
- une connexion USB qui permet la programmation du microcontrôleur ainsi que l'alimentation de la carte ;
- un connecteur d'alimentation jack (nécessaire pour alimenter la carte si le cordon USB est déconnecté après programmation, pour des applications nomades), une tension comprise entre 7 et 12 V est requise.

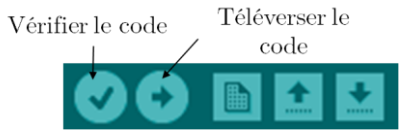
Cette carte met à notre disposition :
- 14 broches numériques d'entrées/sorties (niveau « 0 » soit une tension de 0 V ou niveau « 1 » soit une tension de 5 V) ;
- 6 entrées analogiques acceptant des valeurs comprises entre 0 V et 5 V ;
- différentes sorties de tension : masse, 5 V.

Signalons qu'il est HORS DE QUESTION de travailler avec des tensions supérieures à 5 V, de faire circuler des courants de plus de quelques mA, de faire tourner un moteur ! Lorsque l'on est confronté à des systèmes nécessitant plus de puissance, il faut alors utiliser une alimentation autonome pour le circuit de puissance, l'Arduino ne gérant que la partie commande.

Elle communique avec l'ordinateur grâce au logiciel « Arduino IDE » (environnement de développement intégré).
Elle se programme en langage non pas Python mais C/C++. En pratique on part rarement d'une page blanche mais on adapte un sketch (= programme) déjà existants. Par ailleurs, il existe un certain nombre de bibliothèques clés en main pour divers composants auxquelles on peut faire appel.
A l'ouverture, la fenêtre se présente sensiblement comme ci-dessous. On remarque qu'un fragment de code est déjà présent. Comme indiqué, on place dans la première partie (« setup » = configuration) les instructions destinées à n'être exécutées qu'une seule fois (chargement de bibliothèques, initialisation de variables, …), et dans la seconde (« loop » = boucle) celles qui seront répétées (lecture au cours du temps de la valeur d'une entrée, …).


La carte étant connectée à l'ordinateur via le câble USB les premiers réglages sont :
- Menu « Outils » puis « Type de carte » : vérifier que « Arduino Uno » est bien coché (cocher sinon) ;
- Vérifier que la carte est bien reconnue en allant dans « Port » du même menu. C'est bien le cas si un port est affiché.

Une fois le script saisi, la procédure d'utilisation de la carte est la suivante :
- Vérification du code (et correction si nécessaire !) ;
- Téléversement du programme dans la carte ;
- La communication bidirectionnelle entre l'ordinateur et la carte se fait alors grâce au « moniteur série » (icône en haut à droite et fenêtre en bas).




Les données alors recueillies dans ce moniteur série si elles sont bien formatées pourront faire l'objet d'un traitement ultérieur une fois exportées dans Excel ou Python.

## II Utilisation

### 1) Premiers essais …

Juste histoire pour l'instant de tester le bon fonctionnement du dispositif …
Modifiez le programme précédent afin d'obtenir le code suivant (sans oublier les « ; » et en respectant la casse, i.e. majuscules et minuscules) :

```
1  void setup() {
2    Serial.begin(9600);
3    Serial.println("Hello world !");
4
5  }
6
7  void loop() {
8    // put your main code here, to run repeatedly:
9
10 }
11
```

Lorsqu'un programme Arduino est exécuté, un objet nommé Serial est automatiquement créé (même si on ne l'utilise pas). Cet objet émule un port série et permet de communiquer de façon bidirectionnelle entre l'ordinateur et la carte Arduino.
On précise (ligne 2) que le port série va communiquer à une vitesse de 9600 bauds puis on demande (ligne 3) d'afficher un message qui apparaît dans le moniteur série. Le baud est l'unité de rapidité de modulation en transmission numérique ; la rapidité de modulation d'un signal, exprimée avec cette unité, est égale à l'inverse de la durée en secondes du plus court élément du signal.

Le moniteur série permet non seulement d'afficher des messages reçus de l'Arduino mais également d'en envoyer.
Ouvrez le fichier « Echo » du dossier « PCSI/Arduino » et exécutez-le.

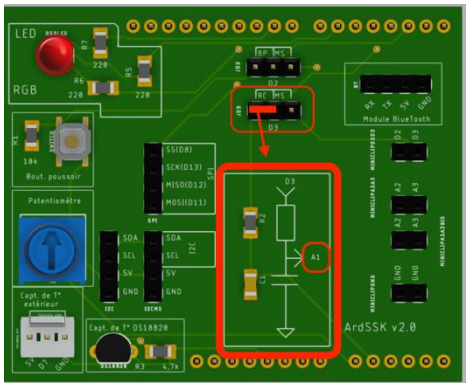
```
1   void setup() {
2     Serial.begin(9600) ;
3     Serial.println("Je suis prêt !") ;
4   }
5
6   void loop() {
7     if (Serial.available())
8     {
9       String message = Serial.readString() ;
10      Serial.println(message) ;
11    }
```

Que se passe-t-il, cette fois, dans la boucle infinie ?
- L'objet Serial scrute en permanence le port série ; si aucune donnée n'arrive, on ne fait rien !
- Si en revanche une donnée arrive sur le port série, on attend la fin de la ligne et on stocke la chaine de caractères reçue dans la variable message (ligne 9) ;
- On renvoie vers l'ordinateur la chaine de caractère en question pour qu'elle s'affiche dans le moniteur série.

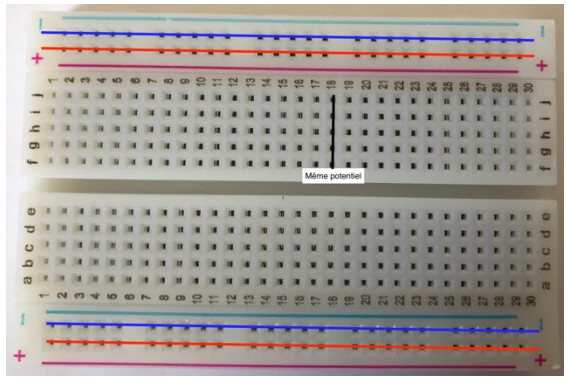
## 2) Commande d'une LED

La LED RGB utilisée est une LED à cathode commune (la borne la plus longue, qui sera reliée à la masse GND). Appliquer un potentiel positif à l'aide des trois broches numériques 6, 9 et 10 (à travers un résistor de protection de résistance 220 Ω) sur l'une des autres bornes de la LED permet d'allumer la couleur correspondante (fonction « digitalWrite (port, état haut ou bas) »).
Ces composants sont ici implantés sur une carte additionnelle (shield = carte d'interface) pluggé sur la carte Arduino.

À défaut, en Tipe par exemple, ces composants seront implantés sur une plaque de prototypage et reliés à la carte par des fils de connexion de couleur. On y trouve deux lignes (-) horizontales, en bleu, non connectées entre elles et dont l'une sera reliée à la masse de l'Arduino, deux lignes (+) horizontales, en rouge, non connectées entre elles et dont l'une sera reliée au + 5V de l'Arduino, et enfin 60 lignes verticales permettant de réaliser le montage. Les connecteurs sont reliés entre eux perpendiculairement à l'axe de la plaque et se trouvent donc au même potentiel.

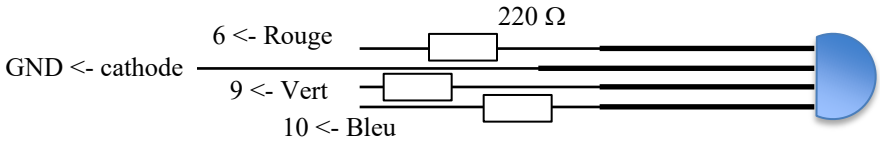



Shield



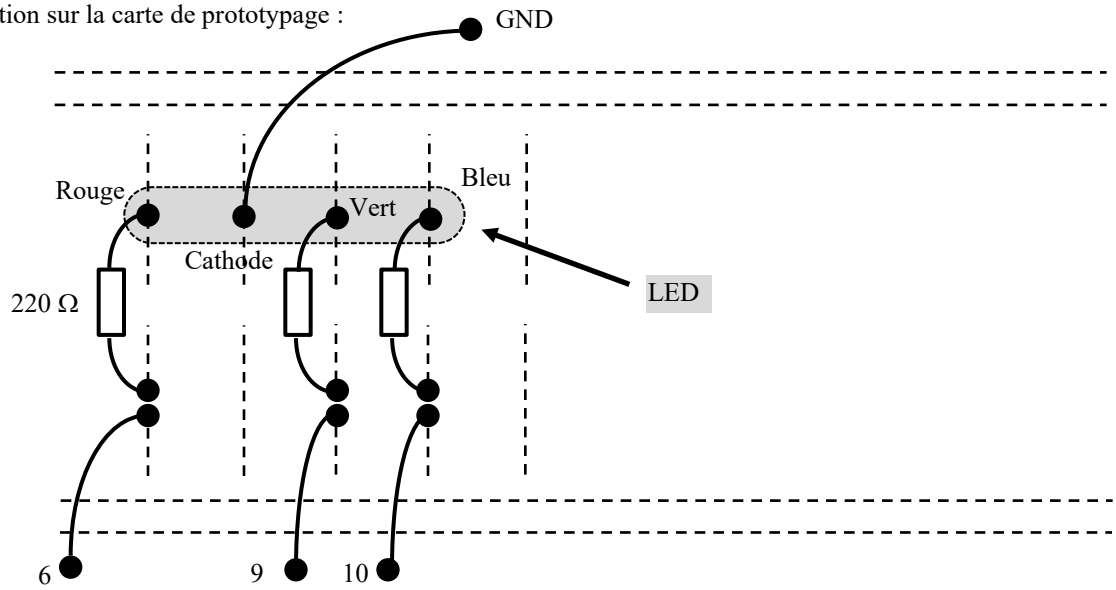

Carte de prototypage

Connexions de la LED :




*Attention, l'absence de résistances de protection entraînerait la destruction immédiate de la diode lors de son branchement.*

Implantation sur la carte de prototypage :




Ouvrez le fichier « RGB » du dossier « PCSI/Arduino » et exécutez-le pour tester le bon fonctionnement de la LED.

```
const int redLEDPin = 6 ;
const int greenLEDPin = 9 ;
const int blueLEDPin = 10 ;

void setup()
{
  digitalWrite(redLEDPin,HIGH) ;
  delay(2000) ;
  digitalWrite(redLEDPin,LOW) ;
  digitalWrite(greenLEDPin,HIGH) ;
  delay(2000) ;
  digitalWrite(greenLEDPin,LOW) ;
  digitalWrite(blueLEDPin,HIGH) ;
  delay(2000) ;
  digitalWrite(blueLEDPin,LOW) ;
}
```

Après avoir défini trois constantes reliant chaque couleur au port correspondant, on allume chaque couleur successivement (« HIGH ») pendant deux mille millisecondes (« delay ») avant de l'éteindre (« LOW »).

**Exercice** : Modifier légèrement le programme pour faire clignoter indéfiniment la LED avec une couleur blanche et une fréquence de 0,2 Hz.

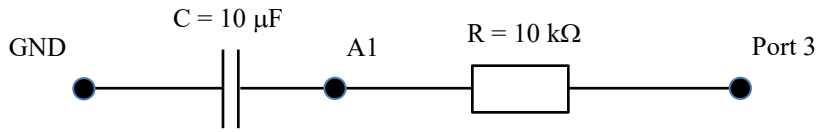
La fonction « analogWrite (port, i) » permet aussi de fixer la valeur de la tension d'un port mais à une valeur comprise entre 0 et 5V, à $i \times \frac{5}{255} V$. Le programme « analogWrite » ci-dessous permet d'augmenter progressivement l'intensité de la couleur verte de la LED à l'aide d'une boucle « for » qui fait croitre par valeurs entières successives la valeur de « i » de 0 à 255, puis de recommencer indéfiniment.

```
const int greenLEDPin = 9 ;

void setup()
{
  pinMode(greenLEDPin,OUTPUT) ;
}

void loop()
{
  for (int i = 0 ; i < 255 ; i++)
  {
    analogWrite(greenLEDPin,i) ;
    delay(20) ;
  }
}
```

**Exercice** : Programmez un feu tricolore …
Les couleurs suivantes doivent se répéter indéfiniment : rouge pendant 4 s, puis vert pendant 4 s, puis orange (red = 255, green = 165, bleu = 0) pendant 2 s.

## 3) Décharge d'un condensateur dans un résistor

Outre la production des signaux de commande précédents, la carte peut aussi procéder à des acquisitions de signaux provenant de capteurs par l'intermédiaire des entrées analogiques.
On se propose ici de réaliser la mesure de la constante de temps d'un circuit RC série, par l'acquisition de la courbe de décharge du condensateur dans un résistor, obtenue en mesurant l'évolution de la tension aux bornes de C au cours du temps.
Le dispositif suivant sera câblé sur la platine de prototypage et relié à la carte :




Le circuit RC est alimenté par la sortie analogique/numérique 3 et le point milieu est connecté à l'entrée analogique A1.
À un facteur multiplicatif près 5/1023, A1 permet de lire la ddp (en V) aux bornes du condensateur.
Le programme est le suivant :

* Setup :
  • On impose sur l'entrée 3 une tension de 5V (digitalWrite(HIGH)) et on attend par exemple 3 secondes (delay(3000)) que le condensateur soit chargé ;
  • À l'instant t = 0 (start = millis()), on applique une tension nulle (digitalWrite(LOW)) sur l'entrée 3 ;
* Loop :
  • On mesure la tension sur l'entrée A1 (valeur = analogRead(A1)) et on envoie les informations sur le port série (println('DATA :TIME :str(millis() - start)) : A1 :valeur')) ;
  • On arrête au bout d'un certain temps, par exemple quand A1 devient inférieure à 1 soit environ 0,1% de sa valeur initiale.

```
#define analogPin   A1
#define commandePin 3
long start = 0 ;
int sensorValue = 0 ;

void setup()
{
  Serial.begin(9600);
  pinMode(commandePin, OUTPUT) ;
  digitalWrite(commandePin, HIGH) ;
  delay(3000) ;
  sensorValue = analogRead(analogPin);
  digitalWrite(commandePin, LOW) ;
  start = millis() ;
}

void loop()

{
int sensorValue = analogRead(analogPin);
  if (sensorValue > 1)
  {
    String msg = "DATA" ;
    msg = msg + ":TIME:" + String(millis()-start) ;
    msg = msg + ":A1:" + String(sensorValue) ;
    Serial.println(msg) ;
  }
}
```

Exécuter le programme « RCdecharge ».

Pour le traitement, les données du moniteur série seront transférées dans la première case d'une feuille Excel par Copier/Coller.
Les données sont maintenant dans la première colonne.

Sélectionnez la première colonne puis, dans le menu « Données », cliquez sur l'item « Convertir ».
La boite de dialogue de l'assistant de conversion s'ouvre.
Sur la première page, choisir l'option « Délimité » indiquant qu'un caractère sépare chacun des champs.
Sur la deuxième, préciser ce caractère, ici « : » et valider.
Les données apparaissent alors séparées dans les premières colonnes.
Se débarrasser des données inutiles.
Les données restantes (temps, valeur de A1) sont maintenant prêtes à être traitées.
Reste donc à faire la représentation graphique …

Déterminer la constante de temps et comparer à la valeur théorique attendue.

**Exercice** : modifier légèrement le programme précédent pour effectuer la charge du condensateur et comparer la constante de temps à la précédente.

**Exercice** : *activité complémentaire optionnelle à ne traiter qu'après la réalisation complète de tout ce qui précède*
On se propose sur la manipulation précédente de la charge du condensateur de mettre en évidence la variabilité de la mesure d'une grandeur physique, en l'occurrence ici la constante de temps $\tau$ du circuit. Le sketch suivant (à compléter) détermine automatiquement, et de manière répétée, cette constante de temps en mesurant le temps nécessaire pour atteindre 63% de la tension maximale.
Lancer le programme une fois complété (aux endroits signalés « A COMPLETER ») afin de recueillir quelques dizaines de valeurs de $\tau$.

```
#define analogPin A1 // définit la broche analogique (A1) de mesure de uc
#define chargePin 3 // définit la broche permettant la charge du condensateur
unsigned long t0 ; // temps initial
unsigned long tau ; // tau constante de temps du circuit RC

void setup() {
  pinMode(chargePin, OUTPUT) ; // définit la broche d'alimentation en sortie
  digitalWrite(chargePin, A COMPLETER) ; // applique 0 V aux bornes du circuit RC
  Serial.begin(9600); // fixe le débit de communication avec le port série

}

void loop()
{
  digitalWrite(chargePin, A COMPLETER) ; // applique 5 V aux bornes du circuit RC
  t0=micros(); // instant initial de la charge du condensateur
  while(analogRead(analogPin)<A COMPLETER){//tant que le niveau lu reste inférieur à 63% du maximum, boucler sans rien faire
}
// Rappel : les valeurs analogRead() sont codées sur 10 bits, entre 0 et 1023

tau=micros()-t0; // durée entre l'instant initial et celui pour lequel la tension atteint 63% du maximum
Serial.print(tau); //affiche tau sur le moniteur série
Serial.print(","); // affiche une virgule après la valeur de tau
Serial.println("microsecondes");// affiche l'unité et saute à la ligne
digitalWrite(chargePin, A COMPLETER); // ramène la tension d'entrée du crcuit RC à 0 V
delay(A COMPLETER); // délai de 3 s pour s'assurer de la décharge totale du condensateur avant une nouvelle mesure
}
```

Exporter les données dans Excel et calculer successivement la moyenne puis l'écart-type.

La copie des données « au vol » pendant les acquisitions étant assez acrobatique, modifier le programme précédent afin que ces dernières s'arrêtent automatiquement au bout de 30 mesures.
Indication : on pourra effectuer un test (« while » ou « if ») sur un entier N correspondant au nombre de mesures (initialement égal à 1 et incrémenté de 1 à chaque nouvelle acquisition).

4) Emploi de « analogRead » (et « analogWrite »)

*Activité complémentaire optionnelle à ne traiter qu'après la réalisation complète de tout ce qui précède.*
*Elle sera réalisée ex nihilo et en complète autonomie. Si nécessaire, des fichiers de corrigés se trouvent sur le serveur.*

Il s'agit ici d'utiliser « analogRead » pour comprendre le principe d'exploitation d'un capteur analogique.
Le port « analog » code la valeur sur 1024 valeurs possibles (sur 10 bits), de 0 à 1023.
Par exemple, on utilise sur le shield un diviseur de tension (potentiomètre rotatif de 10 kΩ) alimenté entre 0 et 5 V et dont le point milieu est connecté à l'entrée analogique A0. On modifie donc la valeur de A0 en tournant le bouton de réglage.

* Première application :
  Utiliser cette valeur lue pour commander l'intensité lumineuse émise par une diode codée sur 256 valeurs (8 bits) de 0 à 255.
  Attention, « analogRead » retourne un entier compris entre 0 et 1023 (10 bits) alors que « analogWrite » attend un entier compris entre 0 et 255 (8 bits).
  Indication : on pourra utiliser la fonction map(entier,min1,max1,min2,max2) qui convertit un entier compris entre min1 et max1 en un entier valeur compris entre min2 et max2.
  Correction : « Eclairage_reglable ».

* Deuxième application :
  Régler la fréquence de clignotement d'une led, en modulant la valeur de « Delay » en exploitant la valeur lue sur A0, à convertir avec un facteur de proportionnalité adéquat afin d'avoir des valeurs adaptées.
  Correction : « Clignotant_reglable ».