Informatique embarquée avec une carte Arduino

Daurelle Anthony
novembre 2014

Stage au LIRMM / université de Montpellier 2

Classe de seconde
Baccalauréat professionnel SEN

Lycée Jean-François Champollion - Lattes Année scolaire 2014-15

Remerciements

Je remercie le directeur du LIRMM, Jean-Claude König, de m'avoir autorisé à y effectuer mon stage.

Je remercie aussi Philippe Reitz d'avoir accepté d'être mon tuteur de stage, et pour tout ce qu'il m'a montré et appris au cours de ce stage.

Ce rapport a été rédigé avec *OpenOffice* ; les schémas électroniques ont été élaborés avec *Fritzing*.

Sommaire

Introduction

J’ai passé mon stage au LIRMM, un laboratoire de recherche en informatique, avec des enseignants et chercheurs. Ce stage s'est déroulé du 10 au 29 novembre 2014.

Dans les différents domaines qu'inclut le BAC PRO SEN figure l'informatique embarquée : c'est dans ce cadre que mon stage s'est déroulé, au travers de montages électroniques avec un Kit Arduino.

Ce rapport est organisé ainsi :

- Description du LIRMM et des personnes que j’ai rencontrées.
- Les activités que j’ai pu réaliser.
- Les informations que j’ai pu avoir, comprendre, et ce que j’en ai tiré.

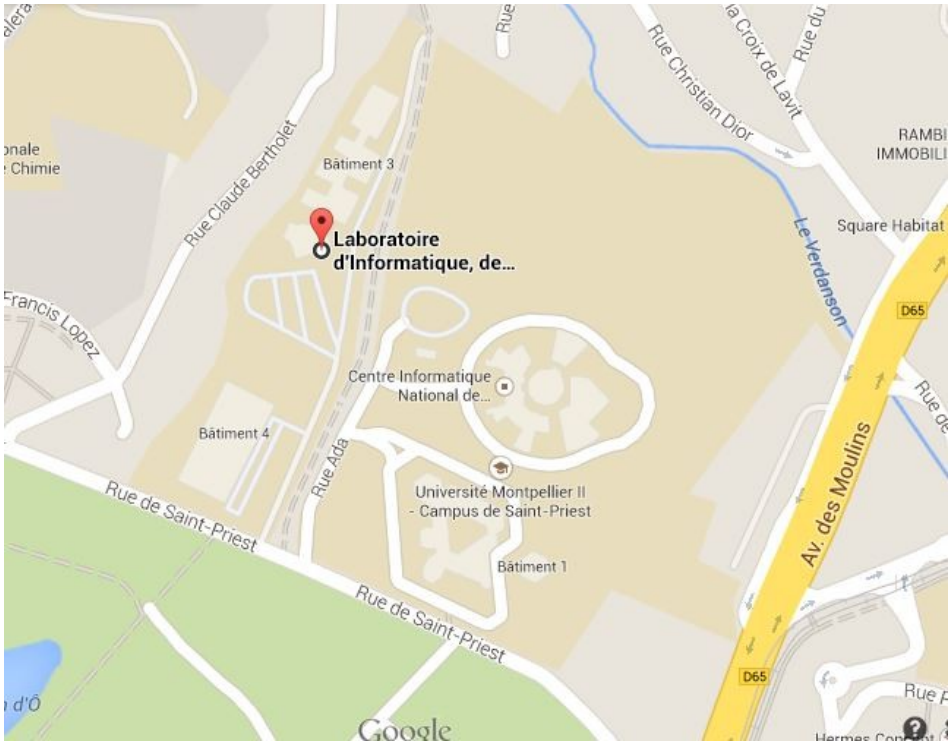
Contexte du stage

Présentation du LIRMM.

Plan des locaux et situation géographique

Le LIRMM se trouve sur le campus scientifique de Saint-Priest, juste à coté du parc du Château d'Ô à Montpellier, célèbre pour ses spectacles du Printemps des Comédiens, comme le montre le plan suivant (fait avec Google Maps) :

Ce campus est composé de 5 bâtiments (le dernier est non visible sur le schéma ci-dessous car il vient juste d'être construit. – il occupera l'espace laissé libre au nord du campus), qui sont tous liés à l'université de Montpellier 2 (faculté des Sciences) ; le LIRMM est le bâtiment 3, au nord-ouest du campus :



Ses principales coordonnées :

Adresse : 161, rue Ada
34090 Montpellier cedex 5
Web : http://www.LIRMM.fr

Secteur d'activité :

Enseignement supérieur et recherche universitaire.

Statut Juridique :

Le LIRMM est une unité mixte de recherche (UMR), qui est gérée à 50% par le CNRS (Centre National de la Recherche Scientifique), un établissement public à caractère scientifique et technologique (EPST), et à 50% par l'université Montpellier 2 (UM2), qui est un établissement public à caractère scientifique, culturel et professionnel (EPSCP). C'est l'un des laboratoires rattachés à l'UM2, qui en compte plusieurs dizaines, organisés par domaines scientifiques ; le LIRMM est dans le domaine « Mathématiques, Informatique, Physique et Systèmes » (MIPS), lequel compte 11 laboratoires différents.

Date de création :

1992

Description plus détaillée :

Quelques pages du site officiel du LIRMM sont reproduites ci-après ; j'ai placé en annexe 3 la plaquette complète.

© Luc Jennepin

Le Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier - LIRMM - est une unité mixte de recherche, dépendant conjointement de l'Université Montpellier 2 et du Centre National de la Recherche Scientifique. Il est situé sur le Campus Saint-Priest de l'UM2.

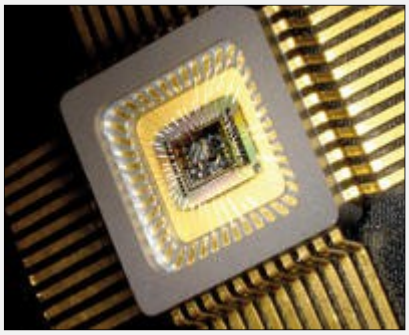
Ses activités de recherche positionnent pleinement le LIRMM au coeur des sciences et technologies de l'information, de la communication et des systèmes.

Ainsi, de l'information aux systèmes, de la technologie à l'humain et aux usages, les activités de recherche du LIRMM concernent : la conception et la vérification de systèmes intégrés, mobiles, communicants, la modélisation de systèmes complexes à base d'agents, les études en algorithmique, bioinformatique, interactions homme-machine, robotique,etc.

Les travaux sont menés dans trois départements scientifiques de recherche, eux-mêmes organisés en « équipes-projet ».

Les thématiques du département **Informatique** s'étendent des frontières des mathématiques à la recherche appliquée : algorithmique des graphes, bioinformatique, cryptographie, réseaux, bases de données et systèmes d'information (intégration de données, fouille de données, maintien de la cohérence), génie logiciel (langages de programmation, objets, composants, modèles), intelligence artificielle (apprentissage, contraintes, représentation des connaissances, systèmes multi-agents), interaction homme-machine (langage naturel, visualisation, web sémantique et e-learning).

Le département **Microélectronique** mène des recherches de pointe dans les domaines de la conception et du test de systèmes intégrés et microsystèmes en mettant l'accent sur les aspects architectures, modélisation et méthodologie.



Enfin, le département **Robotique** s'intéresse à des problèmes de synthèse, de supervision et de gestion de systèmes dynamiques complexes (robots, interface robot/vivant), mais aussi de navigation, localisation et de pilotage de véhicules autonomes présents ou distants, ou encore d'autres portant sur l'analyse, le codage et le traitement d'images. Une des particularités du LIRMM est que théorie, outils, expérimentations et applications sont présents dans tous ses domaines de compétence scientifique.

Les recherches menées au LIRMM trouvent généralement une finalisation dans des domaines applicatifs aussi divers que la biologie, la chimie, les télécommunications, la santé, l'environnement...et dans les domaines propres du laboratoire : l'informatique, l'électronique et l'automatique.

La mission du LIRMM est donc de « produire » :

- des « connaissances » (en moyenne 300 publications d'audience internationale chaque année)
- des « chercheurs » directement (docteurs, post doc) ou indirectement (participation LMD)
- des « objets » matériels et/ou logiciels prototypes
- de l'activité économique : partenariats industriels, création d'entreprises innovantes
- de l'animation scientifique à l'échelle nationale mais aussi internationale.

Les salariés et le chef d'établissement

En novembre 2014, environ 430 personnes travaillent au LIRMM. Un organigramme du laboratoire est visible en annexe 2 ; il est dirigé par Jean-Claude König, enseignant-chercheur en informatique ; le directeur et son équipe sont élus par les personnels tous les 4 ans.

Mon tuteur de stage

Philippe Reitz est enseignant-chercheur (plus précisément maître de conférences, le premier grade dans ce corps de l'éducation nationale, qui en compte deux ; le second grade est professeur des universités) en informatique depuis 1992, et travaille dans l'équipe MAREL du département informatique. Son domaine de recherche porte sur les langages de programmation dédiés aux machines massivement parallèles, appliqués en particulier à la simulation de systèmes naturels, comme par exemple la croissance de cultures (en lien avec le CIRAD, centre de recherche en agronomie). Âgé de 52 ans, son salaire de base est d'environ 3200€ nets par mois, auquel il faut ajouter des primes et des indemnités d'heures de cours

complémentaires (dispensés à l'UM2 ou au CNAM de Montpellier).

Les étapes principales de sa formation sont :

- 1981 : BAC E.
- 1986 : Diplôme d'ingénieur en automatique de l'ESIEE.
- 1992 : Doctorat en informatique de l'université de Montpellier 2.

Ses activités se classent en trois catégories :

- L'enseignement, où il enseigne une matière face à un groupe d'étudiants ; il enseigne plusieurs matières différentes (outils de bureautique, bases de données, programmation dans divers langages, mathématiques de l'informatique, ...), et est l'un des enseignants d'informatique du département GEA (Gestion des Entreprises et Administrations) à l'IUT de Nîmes, l'une des composantes délocalisées de l'UM2. L'IUT diplôme principalement les étudiants avec un DUT (Bac+2) ou une licence professionnelle (Bac+3) : ce sont donc des diplômes de niveau L (Licence), autrefois appelé 1er cycle universitaire.
- La recherche, qu'il pratique au LIRMM, où il participe à des réunions de travail ou de réflexion sur différents sujets, et où il rédige des documents : articles scientifiques, notes personnelles ; il lit beaucoup de documents (articles, livres, thèses de doctorat), les bibliothèques du LIRMM et de l'UM2 permettant d'accéder à des documents du monde entier (dont beaucoup de revues et livres électroniques accessibles en ligne) ; en informatique, ces documents sont souvent en anglais, langue des échanges internationaux dans le monde scientifique.
- L'administration, où il gère des informations ou matériels qui n'ont pas de rapport direct avec les activités des deux catégories précédentes : gestion de la logistique informatique du département GEA à Nîmes, recrutement des nouveaux étudiants ou des nouveaux enseignants, projets d'ouverture de nouveaux enseignements ou diplômes, maintenance de sites Web (pédagogie ou recherche), exploitation de logiciels administratifs de l'UM2 (ENT, emplois du temps, structures des enseignements ou des diplômes)…

Mes activités

Tout au long de mon stage, j'ai réalisé, avec l'aide de Philippe Reitz, des montages électroniques pilotés par une carte Arduino, laquelle est composée d'un micro-contrôleur programmable en C++ depuis un ordinateur connecté via un câble USB.

Nous ne pourrons pas exposer dans ce rapport tous les montages réalisés par manque de place ; néanmoins nous présentons les montages qui nous semblent les plus exemplaires. Tous les montages étudiés sont présentés sur ce site :

`http://www.lirmm.fr/~reitz/arduino/`

Le kit Arduino utilisé est décrit en annexe 4 ; sont présentées les caractéristiques de la carte Arduino Uno R3 exploitée, en particulier quelques informations sur le micro-contrôleur qui l'équipe.

Quelques bases d'électroniques sont exposées dans l'annexe 5 ; l'environnement de programmation (IDE) est décrit à l'annexe 6, ainsi que la structure typique d'un programme C++ définissant le comportement de l'Arduino.

Montages de base réalisés

La plupart des montages réalisés sont ceux du Kit Creator de l'association Fritzing.

Dans la suite, nous supposons définies les fonctions suivantes :

- `complement(int niveau)`, avec `niveau` valant `HIGH` ou `LOW`, renvoie le complément du niveau : `complement(HIGH)` vaut `LOW`, et `complement(LOW)` vaut `HIGH`.

Montage n°1 : une LED clignotante

Ce montage consiste à piloter une LED avec la carte Arduino pour la faire clignoter. La LED est branchée sur la sortie digitale D13, et commandée en tout ou rien (0-1).

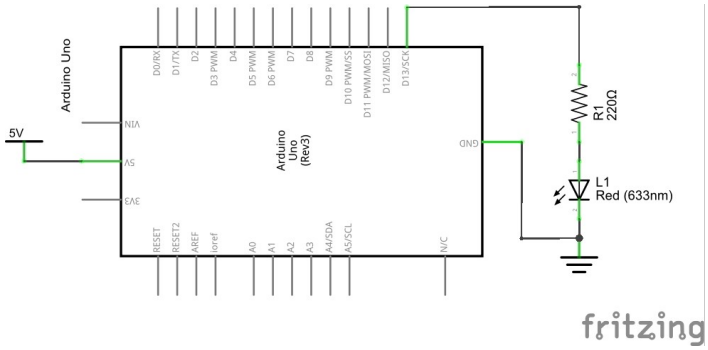


Schéma électronique n°1

a) gestion du clignotement avec la fonction `delay` : cette première solution s'appuie sur la fonction prédéfinie `delay`, qui permet au programme d'attendre un temps spécifié en paramètre, exprimé en millisecondes. Le programme est composé de sa partie initialisation :

```
const int pinL1 = 13;     // broche associée à la LED
const int delaiL1 = 500; // délai d'attente de maintien de la LED ; demi-période du clignotement

void setup() {
  pinMode(pinL1, OUTPUT);     // broche programmée en sortie
  digitalWrite(pinL1, LOW); // LED placée à l'état 0, donc éteinte
}
```

Cette partie définit la broche D13 en sortie digitale puis éteint la LED ; le délai de clignotement étant défini par une constante (`délaiL1`), il est facile de le changer.

La partie active du programme est :

```
void loop() {
  digitalWrite(pinL1, LOW); // la LED passe à l'état éteint
  delay(delaiL1);           // délai d'attente
  digitalWrite(pinL1, HIGH);// la LED passe à l'état allumé
  delay(delaiL1);           // délai d'attente
}
```

Cette partie éteint la LED, attend un certain délai T, allume la LED, puis attend encore ce même délai T ; la fonction `loop` s'exécutant à l'infini, la LED va donc clignoter avec une période $2T$ (soit une fréquence de $\frac{1}{2T}$). Dans le code, ce délai vaut 500ms, la période de clignotement est donc de 1 seconde, soit une fréquence de 1Hz.

b) programme sans la fonction `delay` : l'inconvénient de la solution précédente est que l'Arduino ne peut rien faire d'autre pendant qu'il exécute sa fonction d'attente ; dans cet exemple, cela ne pose pas de problème, mais s'il doit gérer d'autres événements en parallèle, cette attente peut être problématique. Il est possible de programmer un délai d'attente sans passer par la fonction `delay` : la partie initialisation devient alors :

```
const int pinL1 = 13;         // broche sur laquelle la LED est branchée
const int delaiL1 = 500;      // délai d'attente du maintien de l'état de la LED
int derniereHorlogeL1 = -1; // valeur de l'horloge lorsque la LED a changé d'état
int etatL1 = LOW;             // état courant de la LED ; ici LED initialement éteinte

void setup() {
  pinMode(pinL1, OUTPUT);       // broche de la LED programmée en sortie
  digitalWrite(pinL1, etatL1); // LED placée à l'état initial, donc éteinte
}
```

La partie active se résume à :

```
void loop() {
  if (derniereHorlogeL1==-1) // la 1ère fois positionne la variable contenant l'horloge
    derniereHorlogeL1 = millis(); // au dernier changement d'état

  int horlogeCouranteL1 = millis(); // consulte la valeur de l'horloge courante
  if (horlogeCouranteL1-derniereHorlogeL1 >= delaiL1) {
    // s'il s'est écoulé plus que le délai d'attente, change l'état de la LED
    derniereHorlogeL1 = horlogeCouranteL1;
    etatL1 = complement(etatL1);
    digitalWrite(pinL1, etatL1);
} }
```

Montage n°2 : une LED commandée par un bouton poussoir

Ce montage est similaire au précédent, excepté que l'allumage ou l'extinction de la LED est commandé par un bouton poussoir (ou interrupteur fugitif) S1 : tant que le bouton est enfoncé, la LED reste allumée ; lorsqu'il est relâché, la LED s'éteint. Nous étudions différents cas de figure :

- dans un premier temps, nous étudions le cas où la LED reste allumée en permanence quand le bouton est enfoncé, et reste éteinte sinon.
- dans un second, nous reprenons le premier cas, mais la LED clignote au lieu de rester allumée : elle clignote quand le bouton est enfoncé, et reste éteinte sinon.
- dans un troisième temps, la LED change d'état quand le bouton est enfoncé puis relâché.

Le schéma électronique est identique dans tous les cas :

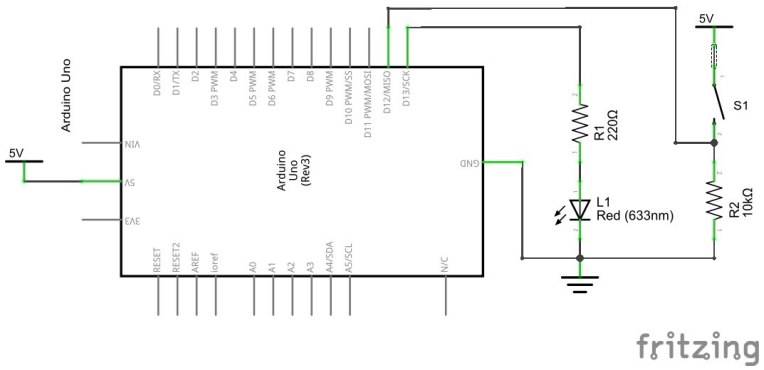


Schéma électronique n°2

a) cas de figure n°1 : l'état de la LED suit celui du bouton poussoir.

La solution la plus simple consiste à définir l'état de la LED comme étant celui de l'état du bouton :

- si le bouton n'est pas enfoncé, la LED est éteinte
- si le bouton est enfoncé, la LED est allumée

Cette gestion se traduit par le programme suivant ; la partie initialisation est :

```
const int pinL1 = 13; // broche associée à la LED
const int pinS1 = 12; // broche associée à l'interrupteur

void setup() {
  pinMode(pinL1, OUTPUT);    // broche de la LED programmée en sortie
  digitalWrite(pinL1, LOW); // LED placée à l'état 0, donc éteinte
  pinMode(pinS1, INPUT);     // broche de l'interrupteur programmée en entrée
}
```

La partie active du programme est :

```
void loop() {
  int etatS1 = digitalRead(pinS1); // récupère l'état de l'interrupteur
  digitalWrite(pinL1, etatS1);     // force la LED dans l'état de l'interrupteur
}
```

b) cas de figure n°2 : identique au cas précédent, excepté que la LED clignote au lieu de rester allumée en permanence quand le bouton poussoir est enfoncé.

Une première solution consiste à mélanger les solutions du montage précédent (gestion du clignotement de la LED) et du cas de figure précédent (contrôle de la LED à partir du bouton) ; sachant que la LED peut être allumée ou éteinte n'importe quand, une solution avec `delay` ne peut être retenue : si l'utilisateur appuie sur le bouton puis le relâche au bout de 200 ou 300ms, le code avec `delay` laisserait la LED allumée 500ms, ce qui ne correspond pas à notre cahier des charges : la LED ne doit clignoter que quand le bouton reste enfoncé.

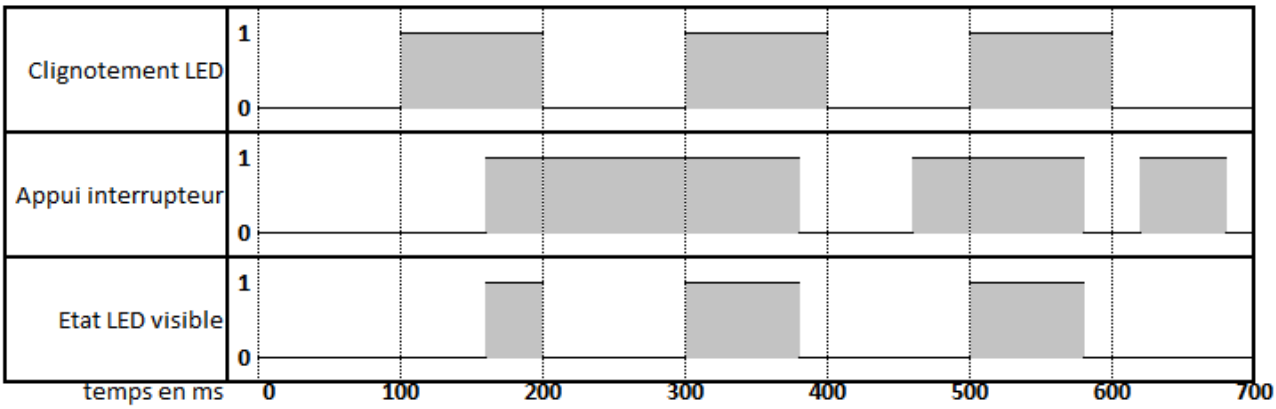
Une première solution est la suivante :

```
const int pinL1 = 13;          // broche sur laquelle la LED est branchée
const int delaiL1 = 500;       // délai d'attente du maintien de l'état de la LED
int derniereHorlogeL1 = -1; // valeur de l'horloge lorsque la LED a changé d'état
int etatL1 = LOW;              // état de la LED ; ici LED initialement éteinte

const int pinS1 = 12;          // broche associée à l'interrupteur

void setup() {
  pinMode(pinL1, OUTPUT);    // broche de la LED programmée en sortie
  digitalWrite(pinL1, LOW); // LED placée à l'état 0, donc éteinte
  pinMode(pinS1, INPUT);     // broche de l'interrupteur programmée en entrée
}
```

La partie active est un mix des deux solutions précédentes :

```
void loop() {
  if (dernieretHorlogeL1==-1) // la 1ère fois positionne la variable contenant l'horloge
    dernieretHorlogeL1 = millis(); // au dernier changement d'état

  int horlogeCouranteL1 = millis(); // consulte la valeur de l'horloge courante
  if (horlogeCouranteL1-dernieretHorlogeL1 >= delaiL1) {
    // s'il s'est écoulé plus que le délai d'attente, change l'état de la LED
    dernieretHorlogeL1 = horlogeCouranteL1;
    etatL1 = complement(etatL1);
  };
  int etatS1 = digitalRead(pinS1);// récupère l'état de l'interrupteur
  if (etatS1 == HIGH)             // si l'interrupteur est enfoncé
    digitalWrite(pinL1, etatL1);  // force la LED dans l'état de clignotement en cours
  else                            // sinon
    digitalWrite(pinL1, LOW);     // force la LED dans l'état 0
}
```

La gestion du clignotement de la LED est indépendante de l'état du bouton : la LED adopte son état de clignotement dès lors que le bouton est enfoncé, sinon reste éteinte ; il est donc possible que la LED reste éteinte bien que le bouton soit enfoncé : il suffit que la LED soit dans son état 0 au niveau clignotement, et que la durée d'activation du bouton soit petite par rapport à la période de clignotement ; voici un exemple d'une telle séquence (la demi-période de la LED est de 100ms) :

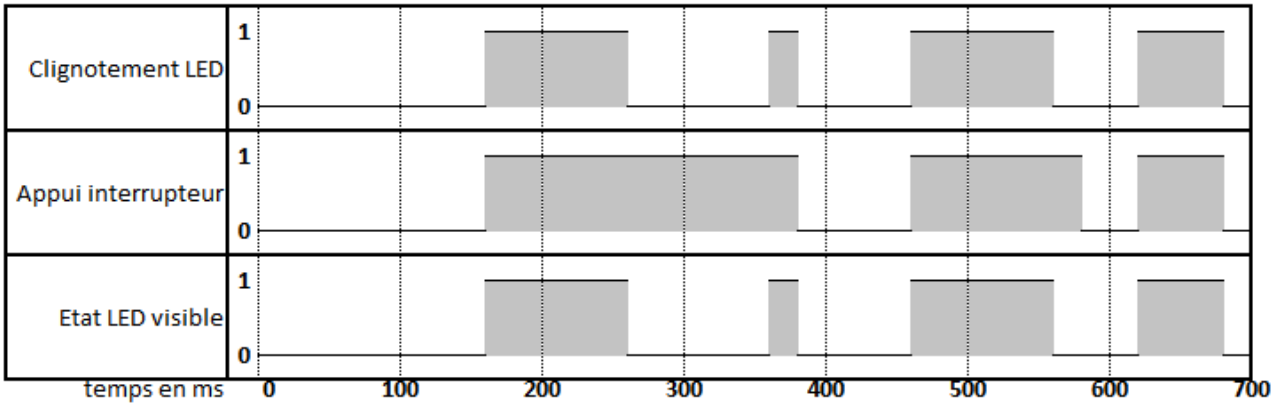


Chronogramme n°1 : état LED et état interrupteur non synchronisés

Nous constatons que :

- lors du 1er appui sur l'interrupteur (entre 180 et 380ms), la LED clignote
- lors du 2ème appui (470 à 580ms), la LED reste allumée le temps de l'appui ; le bouton a été actionné pendant qu'elle était à l'état allumé (courbe supérieure)
- lors du 3ème appui (620 à 690ms), la LED ne s'allume pas ; le bouton a été actionné pendant qu'elle était à l'état éteint.

Nous pourrions souhaiter que l'état de clignotement de la LED soit synchronisé avec l'appui sur le bouton poussoir : dès l'appui sur le bouton, la LED démarre son clignotement à l'état haut ; autrement dit, lorsque l'état du bouton passe de l'état 0 à l'état 1 (au front montant du changement d'état), alors la LED passe de l'état 0 à l'état 1 (front montant du changement d'état de la LED, synchrone avec celui du bouton), démarrant sa période clignotement ; pour le chronogramme précédent, cela se traduirait ainsi :



Chronogramme n°2 : état LED et état interrupteur synchronisés

Nous ne développons pas le code associé ; il est accessible sur le site Web associé à ce stage.

c) cas de figure n°3 : la LED clignote quand on appuie puis relâche une première fois le bouton poussoir ; ce clignotement s'arrête dès que le bouton est à nouveau actionné.

Comme dans le cas de figure précédent, un mix des solutions des montages précédents semble adapté : il suffit de mémoriser dans quel état doit être la LED (doit-elle restée allumée, donc clignotante, ou rester éteinte), cet état basculant quand le bouton est actionné. Comme précédemment, la fonction `delay` ne peut pas être exploitée, puisque le clignotement de la LED est complètement conditionné par les actions sur le bouton, qui peuvent survenir n'importe quand.

Cette stratégie nous conduit au code suivant :

- pour l'initialisation :

```
const int pinL1 = 13;           // broche sur laquelle la LED est branchée
const int delaiL1 = 500;        // délai d'attente du maintien de l'état de la LED
int derniereHorlogeL1 = -1;     // valeur de l'horloge lorsque la LED a changé d'état
int etatL1 = LOW;               // état initial de la LED ; ici LED éteinte
int etatClignotementL1;         // état de clignotement de la LED

const int pinS1 = 12;           // broche associée à l'interrupteur
int dernierEtatS1 = LOW;        // état précédent de l'interrupteur

void setup() {
  pinMode(pinL1, OUTPUT);       // broche de la LED programmée en sortie
  digitalWrite(pinL1, etatL1);  // LED placée dans son état initial
  pinMode(pinS1, INPUT);        // broche de l'interrupteur programmée en entrée
}
```

- pour la partie active :

```
void loop() {
  int etatS1 = digitalRead(pinS1); // lecture de l'état de l'interrupteur
  if (etatS1 != dernierEtatS1) {   // si l'interrupteur a changé d'état
    etatL1 = complement(etatL1);   // inverse l'état de la LED
    dernierEtatS1 = etatS1;        // met à jour la variable d'état
    dernierehorlogeL1 = -1;        // signale qu'il faudra initialiser la variable horloge
  };
  if (etatL1 == HIGH) { // si la LED doit être allumée
    if (dernierehorlogeL1==-1) // au premier passage  positionne la variable horloge
      dernierehorlogeL1 = millis(); // au dernier changement d'état

    int horlogeCouranteL1 = millis(); // consulte la valeur de l'horloge courante
    if (horlogeCouranteL1-dernierehorlogeL1 >= delaiL1) {
      // s'il s'est écoulé plus que le délai d'attente, change l'état de clignotement de la LED
      dernierehorlogeL1 = horlogeCouranteL1;
```

```
      etatClignotementL1 = complement(etatClignotementL1);
    }
  };
  if (etatL1 == HIGH)              // si la LED doit être allumée
    digitalWrite(pinL1, etatClignotementL1);
                                   // force la LED dans l'état de clignotement en cours
  else                             // sinon
    digitalWrite(pinL1, LOW);      // force la LED dans l'état 0
}
```

Cette solution est opérationnelle, mais met en évidence un phénomène gênant : lorsqu'un interrupteur change d'état, les deux éléments qui entrent en contact rebondissent l'un contre l'autre avant de se stabiliser ; ce phénomène dit du *rebond* dure un court instant (de l'ordre de la milliseconde), et est illustré par le diagramme suivant (tiré de l'article *switch* de Wikipedia, section *contact bounce*) :

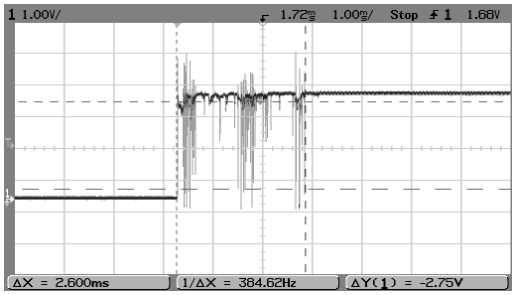


Diagramme montrant l'effet rebond d'un interrupteur

Comme le programme de l'Arduino est exécuté en boucle à une fréquence plus rapide que 1kHz, la lecture de l'état de l'interrupteur, quand il commence à rebondir, alterne entre 0 et 1 rapidement avant que l'état se stabilise à 1 ; sans traitement, le programme considère donc que l'interrupteur a été actionné plusieurs fois, alors qu'un seul appui a eu lieu effectivement.

La correction dans le programme consiste donc à :

1. conserver l'état précédent de l'interrupteur
2. lire l'état courant de l'interrupteur ; si l'interrupteur est passé de 0 à 1, c'est qu'il a été activé ; on fait passer à 1 l'état de l'interrupteur, puis on ignore tous ses changements d'état pendant un certain délai (le temps que les rebonds cessent)
3. on lit à nouveau l'état de l'interrupteur (qui est censé être stabilisé) : si l'interrupteur est resté dans l'état 1, on considère qu'il a été activé, sinon qu'il ne l'a pas été.

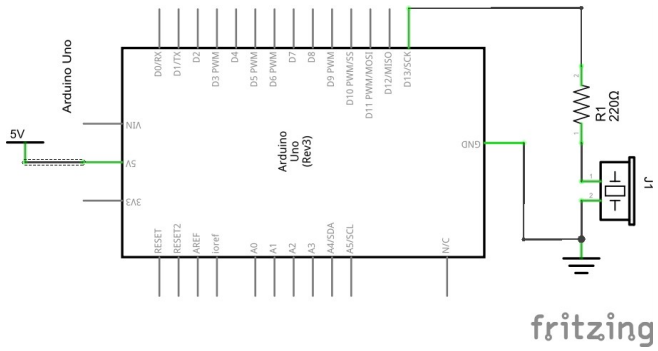
Le code associé est visible sur le site Web.

Montage n°3 : Commande d'un buzzer :

L'intérêt de ce montage est de montrer qu'un Arduino peut piloter en fréquence un buzzer piézoélectrique ; ce pilotage nécessite un changement d'échelle : nous passons du monde des millisecondes à celui des microsecondes, la plus petite échelle de temps que notre Arduino sache traiter.

Le but de ce montage est de générer une note de musique, dont la fréquence est donnée (les sons audibles par l'homme ont une fréquence de quelques dizaines de Hertz à quelques dizaines de kilo Hertz, soit une période de quelques dixièmes de seconde à quelques dixièmes de milliseconde, respectivement) ; on pourrait imaginer un choix dynamique de la fréquence, par exemple via un potentiomètre.

Le schéma électronique de ce montage est le suivant :



Montage n°3 : un buzzer piloté en fréquence par l'Arduino

- pour l'initialisation :

```
const int brocheBuzzer = 13;  // broche pilotant le buzzer
long periodeBuzzer;           // demi-période du buzzer en microseconde
double frequenceNote = 880;   // fréquence de la note souhaitée, ici un « la » de l'octave 4 (la4)
int etatBuzzer = LOW;         // état du buzzer, initialement à 0

void setup() {
  pinMode(brocheBuzzer, OUTPUT);           // broche du buzzer en sortie
  digitalWrite(brocheBuzzer, etatBuzzer); // initialisation de l'état du buzzer
  periodeBuzzer = 500000./frequenceNote;  // conversion note → demi-période buzzer
}
```

Si la fréquence de la note est *f*, sa période est $P=\frac{1}{f}$; comme le buzzer est complètement commandé par l'Arduino, il doit donc être à 1 pendant ½*P*, et à 0 pendant ½*P* ; il change donc d'état chaque demi-période ; il existe dans Arduino une fonction d'attente autre que `delay`, appelée `delayMicroseconds`, dont le temps d'attente est exprimé en microseconde.

- pour la partie active :

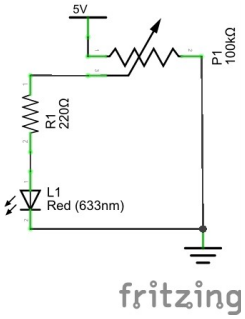
```
void loop() {
  digitalWrite(brocheBuzzer, etatBuzzer);
  delayMicroseconds(periodeBuzzer);
  etatBuzzer = complement(etatBuzzer);
}
```

Montage n°4 : Variation de l'intensité d'une LED via un potentiomètre

Le problème posé dans ce montage est de faire varier l'intensité lumineuse d'une LED via un potentiomètre de contrôle ; il y a plusieurs façons de gérer ce problème, par exemple :

- soit sans Arduino, l'intensité lumineuse la LED étant contrôlée directement par le courant continu qui la traverse.
- soit la LED est pilotée par une sortie digitale de l'Arduino, et son intensité lumineuse est alors fonction du courant moyen qui la traverse, via une technique appelée *modulation en largeur d'impulsion* (MLI en français, ou PWM en anglais, pour *Pulse Width Modulation*).

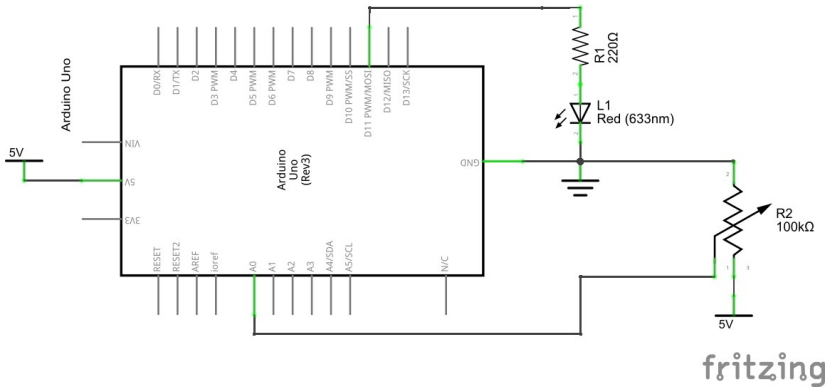
a) commande analogique ; le schéma électronique de ce montage est le suivant :



Montage n°4a : une LED dont l'intensité est contrôlée de façon analogique

Ce montage est le plus simple, et ne nécessite aucun composant additionnel : il suffit d'appliquer à la LED la tension obtenue au niveau du potentiomètre.

b) commande digitale par MLI; le schéma électronique de ce montage est le suivant :

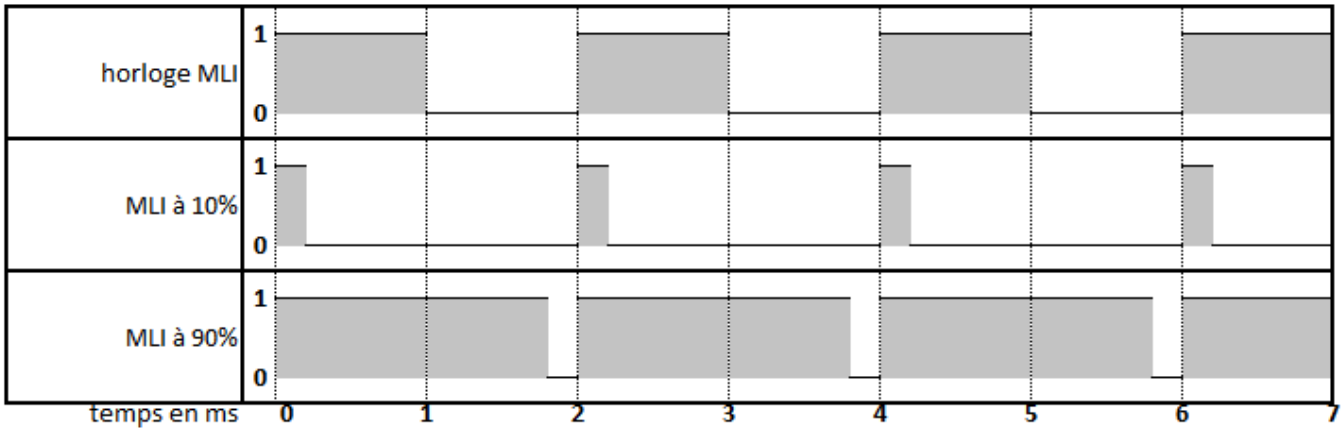


Montage n°4b : une LED dont l'intensité est contrôlée par MLI

Dans ce montage, on commande la LED via une broche digitale, qui fonctionne en principe en tout ou rien, et pourtant nous sommes capables de modifier son intensité : l'astuce réside dans la technique dite de modulation en largeur d'impulsion (MLI, ou en anglais PWM, pour *Pulse Width Modulation*).

Le principe est le suivant :

- un signal binaire est envoyé sur une broche avec une période T (1ère ligne du chronogramme ci-dessous)
- sur chaque période, le signal reste à l'état 1 pendant n % de la période, puis passe à l'état 0 le reste du temps, soit 1-n % ; si l'on s'intéresse à la valeur moyenne du signal sur un temps long (très grand devant la période T), on constate que cette valeur moyenne est exactement n
- si le niveau 1 du signal correspond à une tension U et que le niveau 0 correspond à une tension nulle, alors avec un n donné (compris entre 0% et 100%), la tension moyenne générée sera égale à $n\,U$.



Chronogramme MLI

Dans le chronogramme ci-dessus, deux cas de figure sont présentés :

- si *n*=10 %, la tension de sortie moyenne envoyée à la LED sera de 0,5V (10 % de 5V) : la LED sera éteinte.
- Si *n*=90 %, la tension de sortie moyenne envoyée à la LED sera de 4,5V (90 % de 5V) : la LED sera allumée, mais pas avec son intensité maximale, atteinte quand *n*=100 %.

Au niveau de l'Arduino, les ports digitaux acceptant le mode MLI sont en nombre limité, et le pourcentage est exprimé comme un entier compris entre 0 et 255 (bornes incluses), 0 étant associé à 0 %, et 255 à 100 %.

Concernant le programme, nous avons :

- pour l'initialisation :

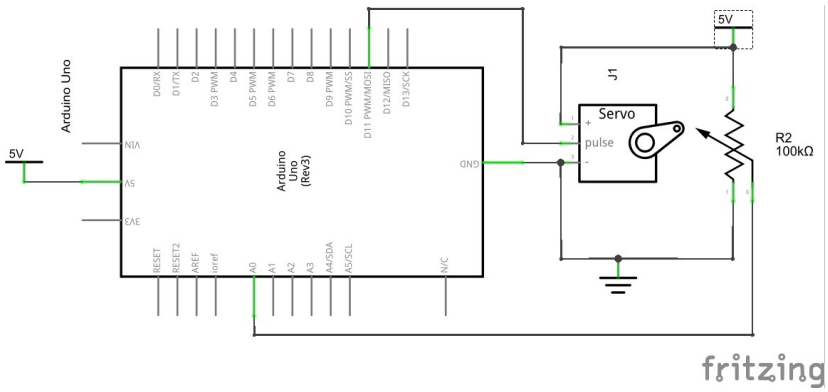
```
const int potentiometre = A0; // broche du potentiomètre
const int LED = 11;           // broche de la LED sur une broche digitale en PWM
int valeurPot = 0;            // valeur associée au potentiomètre, de 0 à 1023
int valeurLED = 0;            // valeur associée à la LED, de 0 à 255

void setup() {
  pinMode(LED, OUTPUT);       // la broche de la LED est programmée en sortie
  analogWrite(LED, valeurLED);// éteint la LED au départ
}
```

- pour la partie active :

```
void loop() {
  valeurPot = analogRead(potentiometre);       // lit la valeur du potentiomètre
  valeurLED = map(valeurPot, 0, 1023, 0, 255);// convertit cette valeur en une tension
  analogWrite(LED, valeurLED);                 // produit cette tension sur la LED
  delay(100);                                  // délai d'attente avant prochain tour
}
```

Montage n°5 : Contrôle d'un servomoteur via un potentiomètre :

Le schéma électronique de ce montage est le suivant :



Montage n°5 : un servomoteur contrôlé par un potentiomètre

Ce montage est similaire au précédent dans sa variante MLI, excepté que le composant en sortie n'est pas une LED mais un servomoteur : ce dispositif permet de définir la position angulaire d'un axe, sachant que l'angle demandé est exprimé par un nombre ; ce type de dispositif est très utilisé dans les jouets télé ou radio commandés, ou encore en robotique.

Dans le montage que nous avons testé, l'angle de rotation de l'axe du servomoteur peut varier de 0 à 160° ; le pilotage d'un servomoteur par un Arduino n'étant pas facile à mettre en œuvre avec les fonctions de base, il faut passer par une bibliothèque dédiée à ce problème : `Servo.h` ; le programme devient :

- pour l'initialisation :

```
#include <Servo.h>

Servo servomoteur;              // création d'un objet C++ gérant le servomoteur
const int brocheServo = 11;     // broche du servomoteur, nécessairement MLI
const int potentiometre = A0; // broche analogique du potentiomètre
int valeurPot;                  // valeur associée au potentiomètre, de 0 à 1023

void setup() {
  servomoteur.attach(brocheServo); // associe l'objet servomoteur à la broche adéquate
}
```

- pour la partie active :

```
void loop() {
  valeurPot = analogRead(potentiometre); // lecture de la valeur du potentiomètre : 0..1023
  int angle = map(valeurPot, 0, 1023, 0, 160); // génère la commande angulaire : 0..160
  servomoteur.write(angle);              // commande le servomoteur
  delay(15);                             // délai pour que le servomoteur réalise la demande
}
```

Autres montages

Notre site Web montre d'autres montages réalisés, dont :

- le kit Creator de Frizing contient une matrice de 8x8 LED, chaque LED pouvant être pilotée indépendamment des autres, mais pas forcément toutes en même temps. Un montage décrit dans le kit permet de simuler un jeu de Ping-Pong, chaque joueur contrôlant la position de sa raquette avec un potentiomètre.
- Lorsque nous avons voulu tester ce jeu la première fois, le schéma de brochage de la matrice de LED ne correspondait pas aux descriptions ; nous avons dû retrouver le bon brochage, et nous avons réalisé un montage pour vérifier que ce brochage était correct (allumage des LED une par une, en balayant d'abord les colonnes, puis les lignes) ; dès lors, le jeu du Ping-Pong a pu effectivement être réalisé.

- Nous avons réalisé un montage avec une photorésistance : il s'agit d'une résistance dont la valeur change en fonction de l'intensité de la lumière qu'elle reçoit ; elle se comporte donc comme un potentiomètre, excepté que la lumière remplace la tige de réglage du potentiomètre.
- Nous avons aussi réalisé des montages permettant de piloter un petit moteur électrique à courant continu ; un tel moteur ayant besoin d'un courant d'environ 600mA, une broche de sortie d'un Arduino ne pouvant en produire qu'au plus 40mA, il faut nécessairement un composant entre les deux pour relayer le faible courant de commande de l'Arduino vers le fort courant commandant le moteur ; ce composant peut être un transistor (dans notre cas un MOS FET) ou un circuit intégré spécialisé (dans notre cas un L293D ; ce composant permet de commander l'inversion du sens de rotation du moteur très simplement, et permet de piloter 4 moteurs en même temps).

Étude de cas :

Mon tuteur m'a posé le problème suivant : supposons que nous souhaitions construire un jouet de type voiture de police ; pourrions-nous exploiter notre kit Arduino pour y piloter le maximum d'éléments possibles, et lesquels ?

En fonction des montages que nous avons réalisés, et des limitations du kit à notre disposition, je lui ai proposé :

- les phares avant (2 LED blanches) et arrière (2 LED rouges) de la voiture sont pilotés par l'Arduino selon la luminosité mesurée par une photorésistance : les phares sont éteints de jour, et allumés la nuit, l'intensité des feux étant inversement proportionnelle à l'intensité de la lumière du jour.
- un gyrophare (LED triple dont seules les couleurs rouge et bleu sont utilisées) peut-être activé ou non via un interrupteur fugitif, une sirène (réalisée avec le buzzer) se faisant entendre, avec les fréquences de la police, lorsque le gyrophare fonctionne (clignotement des couleurs bleu et rouge).
- un moteur actionne les roues arrière ; la vitesse est contrôlée par un potentiomètre (de l'arrêt à la vitesse maximale), et le sens de rotation pouvant être choisi après appui sur un interrupteur fugitif.
- un servomoteur permet de commander la direction des roues avant, l'angle de direction étant asservi à un potentiomètre.

Nous avons écrit le programme permettant de piloter tous ces éléments ; il est disponible sur la page Web associée au rapport.

Conclusion

Mon avis sur ce stage est super, je veux dire par là que j'ai bien apprécié l'environnement, les activités, je ne me suis pas ennuyé, et j'ai fait quelque chose qui m'intéresse réellement.

Les points positifs :

Les différents logiciels que mon tuteur m'a conseillé d'installer pour exploiter au mieux mon ordinateur, en particulier ceux liés à la programmation, avec lesquels on peut '' jouer '' (pour mon tuteur, il n'y a qu'une façon d'apprendre la programmation par soi-même : c'est d'écrire de nombreux programmes que l'on teste et re-teste jusqu'à ce qu'ils fonctionnent). Il m'a aussi montré différents petits logiciels pratiques pour l'utilisation de ma machine, qui rendent certains accès beaucoup plus simples, ou qui rajoutent des fonctionnalités, même au niveau de la sécurité ; et il m'a donc montré comme cela fonctionne en général.

Il y a aussi une ambiance très sympa qui règne dans cette entreprise, aucune personne ne se moque de quelqu'un d'autre, ils ont pourtant tous des styles différents (caractères, vêtements…), et sont de nationalités différentes : syriens, chinois, brésiliens, hindous, japonais, ... Ce qui fait qu'on a donc du plaisir à se rendre dans ces locaux et à travailler. De plus, les murs ainsi que la façade sont entretenus, et donc cela fait du LIRMM un beau bâtiment tout de même.

Les points négatifs :

Les bureaux sont petits (voire très petits) et pourtant, c'est pour 2 ou 3 personnes plus souvent. Personnellement, j'y mettrais juste une personne.

Les ordinateurs (souvent des DELL ou HP, suivant lequel des deux décroche les contrats suite aux appels d'offre - marchés publics) sont de bas niveau, je veux dire par là qu'ils n'ont pas beaucoup de puissance, de mémoire vive.. et donc largement pas assez pour un informaticien ; par contre, tous ces ordinateurs sont connectés en réseau, y compris à des ordinateurs de grosse puissance (serveurs de fichiers ou de calcul), enfermés dans une salle spéciale sécurisée, dans laquelle n'entrent que quelques techniciens autorisés ; certains chercheurs achètent parfois leur propre PC, avec une assez grosse puissance, et dont le prix tourne autour des 1200/1500 euros pour un PC portable.

Sinon, aucun autre point négatif à remarquer spécialement.

Pourquoi avoir choisi cette entreprise

Déjà car c'est un métier que j'envisage plus tard, ou en tout cas dans la même branche/secteur. Et aussi car c'est une connaissance de mes parents, et ils m'ont conseillé cela, ils ont demandé à Philippe, qui a accepté.

Qu'ai je appris

J'ai appris plusieurs choses lors de ce stage.

J'ai déjà appris les bases de la programmation, c'est-à-dire les différents langages qui existent, et aussi savoir vraiment comment fonctionne un ordinateur, savoir qui fait quoi, qui donne des ordres par rapport à la mémoire vive, disque dur etc..

J'ai aussi découvert les bases de l'électronique, et appris à réaliser quelques montages avec les plaques d'expérimentation : c'est vraiment très facile de tester des choses, mais certaines erreurs peuvent être graves : des composants peuvent griller… Durant mon stage, aucun composant n'a grillé, même si, parfois, mon tuteur a eu quelques frayeurs.

Cela m'a beaucoup intéressé, car pour le moment en tout cas, j'adore la programmation, elle me fascine.

Les différentes difficultés que j'ai rencontrées :

Au niveau des relations humaines, aucune car ils sont tous très ouverts d'esprit, et donc aucune moquerie, ou aucune mauvaise entente particulière. Ils sont tous dans la recherche, et ils s'investissent beaucoup dans leur travail.

Au niveau des connaissances professionnelles, là par contre, il faut être assez ‘’ pointu ‘’ car si on se trompe dans une programmation, ou dans le raisonnement qui va nous mener à ce programme, alors là, il n y a plus qu’à recommencer.

Sinon, il faut savoir tout de même bien les différents langages et les bases en particulier, ainsi que savoir raisonner et avoir beaucoup de logique, car beaucoup de problèmes pourtant vus comme simples, ont en fait beaucoup plus de difficultés, et pas toujours de solutions.

Par rapport à la connaissance générale, ceux qui sont admis ou qui font leurs études au LIRMM ont presque tous fait un doctorat (Bac +8) ; ils ont donc une bonne connaissance générale.

L’idée du monde du travail

Au niveau des relations entre les personnes, oui, car je n’avais pas conscience qu’on pouvait avoir un esprit aussi ouvert et donc avoir une très bonne entente ; on voit que beaucoup d’entre eux (voire tous) ont fait de longues études, et ont l’air vraiment passionnés par ce qu’ils font.

Pour l’organisation du travail aussi, car mon tuteur par exemple à un emploi du temps très libre ; il donne des cours à l’IUT de Nîmes, et au CNAM sinon, il travaille parfois chez lui sur différentes recherches qu’on lui demande de faire.

Il a donc un emploi du temps très libre qui lui permet de travailler quand il le souhaite, en particulier chez lui assez souvent (par exemple pendant les vacances scolaires). Cette liberté pourrait nous faire craindre que beaucoup abusent de ce système pour être très souvent en vacances ou en repos. Mon tuteur, bien qu’il ne nie pas que cela puisse se produire, mais de façon très marginale, estime que ce qui empêche les abus est que les chercheurs sont très motivés et passionnés par leur travail : ils ne comptent pas leur temps.

Aux différents métiers, non pas spécialement, je savais déjà à peu près les différents métiers qu’il peut y avoir en relation avec cette branche (au niveau de la robotique etc…).

Elle a par contre changé mon avis personnel sur le monde du travail :

Du ‘’ Je me lève tôt le matin, je pars bosser et je reviens vers 19h sans aucun plaisir ‘’. J’ai appris qu’on pouvait réellement avoir du plaisir à travailler à condition d’avoir le travail et le grade qu’on souhaite.

Mes projets dans le futur

Oui, cela m’a éclairé d’une certaine façon même si le métier que je veux faire reste indécis, vu qu’il y a beaucoup de métiers différents dans cette catégorie.

Bibliographie

Sites de référence sur Arduino

- http://arduino.cc (site visité en Novembre 2014)

  Site de référence sur les différents kits Arduino ; tous les schémas électronique des cartes s'y trouvent ; l'environnement de développement y est également téléchargeable.

- http://fritzing.org (site visité en Novembre 2014)

  Site offrant un logiciel de dessin et de conception de circuits de montages électroniques ; ce logiciel contient en particulier la description de tous les montages du kit Creator exploité dans mon stage.

Références de quelques livres

- **AFFAGARD Bruno, GÉRIDAN Jean-Michel, LAFARGUE Jean-Noël** : *Projets créatifs avec Arduino* ; éditions Pearson, 2014. ISBN 978-2-7440-2617-1
- **BANZI Massimo** : *Démarrez avec Arduino* ; éditions Dunod, collection ETSF, 2013. ISBN 978-2-10-070152-0.
- **BARTMANN Erik** : *Le Grand Livre d'Arduino* ; éditions Eyrolles, collection Serial Markers, 2014. ISBN 978-2-212-13701-9.
- **NUSSEY John** : *Arduino pour les nuls* ; éditions First, collection Pour les nuls, 2014. ISBN 978-2-7540-6429-3

Quelques liens utiles

- http://fr.wikipedia.org/wiki/Arduino (Site visité en Novembre 2014)

  Ce site donne une définition synthétique de ce qu'est une carte Arduino

- http://f-leb.developpez.com/tutoriels/arduino/univers_arduino/part1/ (Site visité en Novembre 2014)

  Ce site est un cours d'introduction aux montages Arduino

- site Wikipedia : http://www.wikipedia.fr

  Ce site bien connu nous a permis d'obtenir des informations générales : organisation de la recherche, les différentes types d'établissements publics, les universités...

Annexes

Annexe 1 : Convention de stage.

Annexe 2 : organigramme du personnel du LIRMM

Annexe 3 : plaquette du LIRMM

Annexe 4 : les cartes Arduino

La carte Arduino que nous avons exploitée est une Arduino Uno R3, la plus commune ; elle coûte environ 20€ ; le micro-contrôleur est un ATmega328 de la société ATMEL, de la famille AVR ; le microprocesseur est de type 8 bits (alors que ceux de nos ordinateurs d'aujourd'hui sont de type 64 bits) ; il dispose d'une horloge de 16MHz (à comparer aux quelques GHz de nos ordinateurs personnels), ce qui lui permet de traiter des tâches courtes devant s'exécuter sans fin durant quelques microsecondes.

Il dispose d'une mémoire flash de 32k$_i$o pour la partie programme, et d'une mémoire vive (RAM) de 2k$_i$o pour stocker les données ; une mémoire morte électriquement effaçable (EEPROM) de 2k$_i$o contient le programme exécuté lorsqu'il est mis sous tension ; dans la carte Arduino, c'est ce programme qui gère la communication USB avec le PC.

Vue du dessus d'une carte Arduino Uno

La carte dispose :

- d'un connecteur USB qui permet de télécharger les programmes conçus sur un PC, et de dialoguer avec lui lorsque le programme de l'Arduino est opérationnel ; un fusible électronique de 500mA protège le PC en cas de mauvaise manipulation.
- de 14 ports numériques programmables en entrée ou en sortie, numérotés de 0 à 13 ; les ports 0 et 1 sont exploités lors des communications avec le PC via la connexion USB (ils ne sont pas exploitables par l'utilisateur si celui-ci exploite ces communications) ; 6 ports peuvent être programmés en MLI (leurs n° sont précédés du signe ~)
- de 6 ports analogiques uniquement programmés en entrée, nommés de A0 à A5 ; ces ports peuvent être transformés en ports numériques ; en entrée, un port analogique a une résolution de 10 bits (la tension en entrée produit un nombre compris entre 0 et 1023 $= 2^{10}$-1). La tension U imposée de l'extérieur au port est convertie en un nombre entier N selon une relation linéaire $N = 1023 \frac{U}{V_{réf}}$, où $V_{réf}$ est une tension de référence, par défaut celle de l'alimentation de l'Arduino, pour nous 5V. Le dispositif intégré au micro-contrôleur capable de faire cette conversion est appelé un *convertisseur analogique/numérique* (CAN).

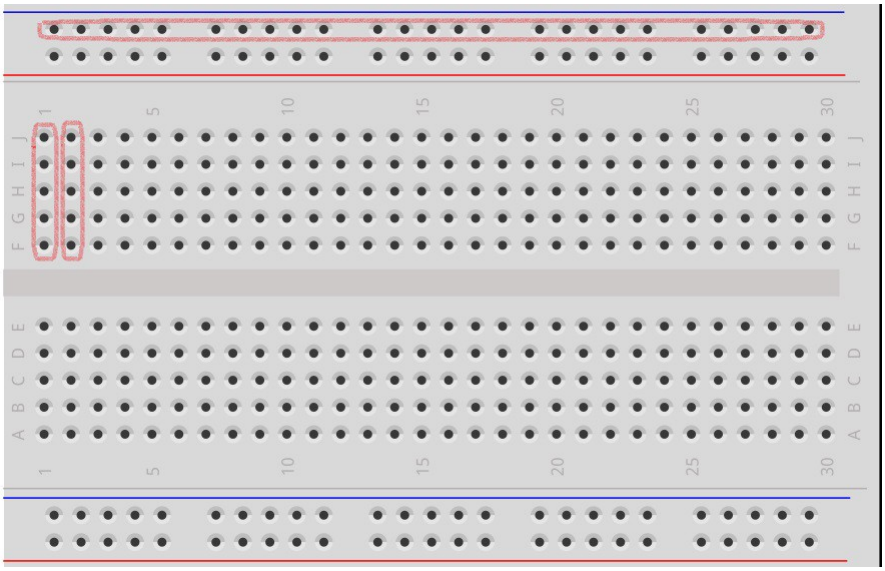
La carte peut être alimentée via sa connexion USB, une pile de 9V ou un système d'alimentation produisant une tension entre 7 et 12V : batterie, chargeur d'appareil électrique, …

Il existe de nombreuses autres cartes Arduino (elles se différencient par la taille, le micro-contrôleur, le nombre de ports numériques ou analogiques) ; par ailleurs, des cartes d'extension (appelées *shield* en anglais) permettent d'augmenter les capacités d'une carte Arduino : ajout d'une connexion Ethernet (liaison filaire ou Wifi), ajout de mémoire (lecteur de cartes mémoire), ajout de connexions spécialisées (par exemple I2C pour la domotique), ajout de relais de puissance (qui permet de piloter des systèmes nécessitant des courants supérieurs à quelques dizaines de milliampères, comme des ampoules ou des moteurs), ...

Annexe 5 : quelques éléments d'électronique

Les plaques d'expérimentation

Tous les montages que nous avons testés ont été réalisés avec des plaques d'expérimentation (ou platines d'essai), similaires à celle-ci :



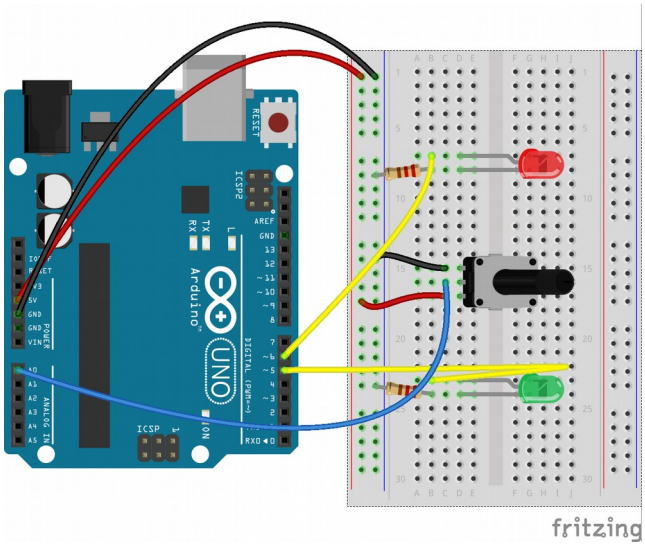
Plaque d'expérimentation

Sur ce schéma, nous avons entouré en rouge tous les trous (points noirs) qui sont connectés électriquement ensemble :

- tous les trous de la ligne horizontale du haut (ceux collés à un trait horizontal bleu) ; sont aussi connectés ensemble tous les trous juste en dessous, le long du trait horizontal rouge.
- les 5 trous placés verticalement dans la partie supérieure gauche ; les 5 trous placés verticalement juste à droite des 5 trous précédents sont eux-même connectés ensemble ; et il en est ainsi pour tous les groupes verticaux de 5 trous voisins.

La demi-partie supérieure de la plaque ainsi que la demi-partie inférieure sont symétriques du point de vue des connexions.

Une telle plaque évite de souder les composants électroniques : il suffit de clipser leurs pattes dans les trous de la plaque d'expérimentation ; voici un exemple de réalisation :



Exemple de montage

Ce montage correspond à une réalisation du schéma électronique suivant :

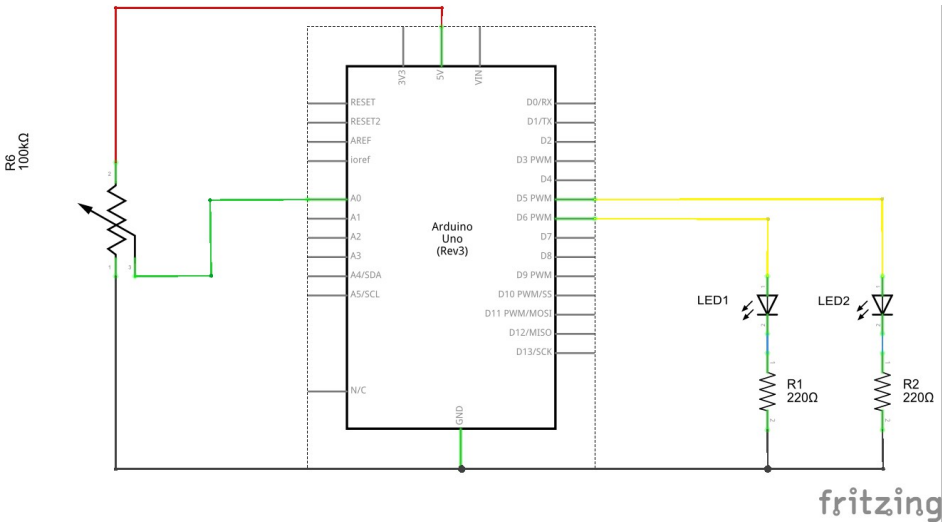


Schéma électronique du montage précédent

Nous avons utilisé dans nos montages différents composants électroniques ; nous en décrivons quelques-uns.

Les résistances

La résistance est l'un des composants les plus simples de l'électronique :

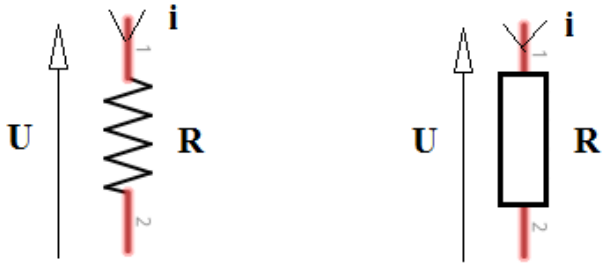


Schéma d'une résistance (convention américaine à gauche, européenne à droite)

Tension U et courant i sont reliés par la loi d'Ohm : $U = R\,i$, où R est la valeur de la résistance exprimée en ohms (symbole : Ω).

Les potentiomètres

Un potentiomètre P est une résistance R variable, schématisé ainsi :

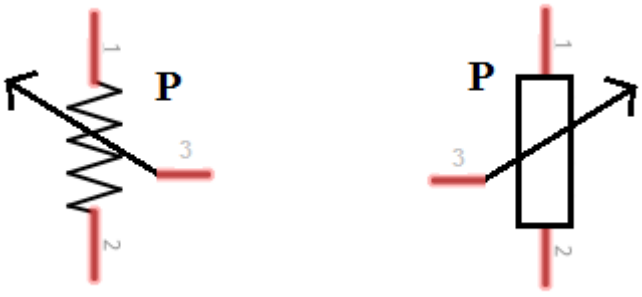


Schéma d'un potentiomètre (convention américaine à gauche, européenne à droite)

La variation se traduit par un coefficient α compris entre 0 % et 100 % correspondant à la position angulaire d'un axe ou la position linéaire d'un curseur pouvant être manipulé par l'utilisateur.

Électroniquement, un potentiomètre est donc équivalent au montage suivant :

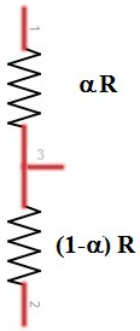
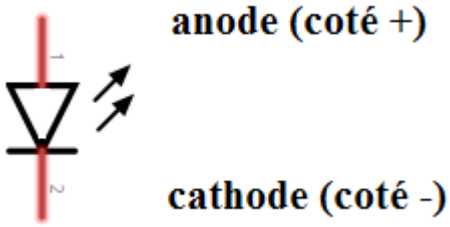


Schéma électronique équivalent d'un potentiomètre

Les LED (ou diodes électro-luminescentes)

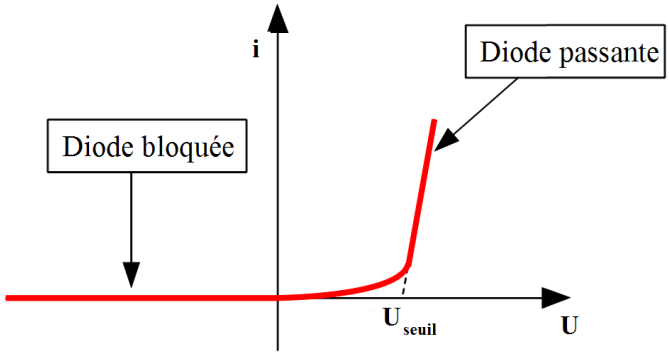
Une LED est une diode qui, quand elle est passante, émet une lumière ; il en existe de différentes couleur : rouge, vert, bleu, jaune, blanc… Nous avons eu l'occasion d'utiliser une LED triple, qui contient en fait 3 LED commandables indépendamment (couleurs rouge, vert, bleu, comme sur une télévision).

Sur un schéma électronique, le symbole d'une LED est :



Symbole d'une LED

La relation entre la tension *U* au borne d'une LED et le courant *i* qui la traverse ressemble à (adapté de Wikipedia) :



Caractéristique d'une LED

Si la tension *U* au borne de la diode dépasse une valeur critique appelée tension de seuil (U_{seuil}), la LED commence à s'éclairer ; le courant ne doit toutefois pas dépasser une valeur limite, au-delà de laquelle la diode grille. Pour une LED rouge, par exemple, le tension de seuil est d'environ 1,9V, et le courant doit être de 20mA pour que la luminosité soit maximale sans détruire la LED.

Plusieurs montages que nous avons construits impliquent des LED ; chaque LED doit toujours être protégée par une résistance afin de garantir que le courant qui la traverse soit de l'ordre de 20mA au maximum :

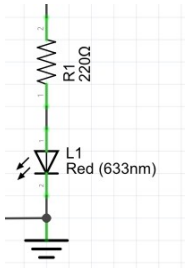


Schéma type de branchement d'une LED

Pourquoi avons-nous choisi une résistance de 220Ω ?

- si la tension appliquée en haut de la résistance est un 0 logique (niveau bas), c'est à dire une tension nulle, alors la LED est dans l'état bloquant, puisque la tension à ses bornes ne peut pas dépasser sa tension de seuil : elle est nulle ; le courant qui la traverse est donc nul aussi
- si la tension appliquée en haut de la résistance est un 1 logique (niveau haut), soit une tension de 5V dans nos montages, alors la LED peut être passante (sa tension est alors de l'ordre de sa tension de seuil, soit 1,9V pour une LED rouge), à condition que son courant soit d'environ 20mA ; si la tension sur la LED est de 1,9V et que le montage est alimenté en 5V, cela veut dire que la tension aux bornes de la résistance est 5-1,9 = 3,1V ; si l'on souhaite un courant de 20mA, la résistance doit avoir une valeur de 3,1 / 0,02, soit 155Ω ; les valeurs des résistances étant normalisées (série E24 pour l'électronique grand public), les valeurs les plus proches de notre valeur théorique sont 150Ω et 180Ω ; comme nous ne disposions que de résistances de 220Ω dans cette échelle de valeurs, c'est cette valeur que nous avons retenue.

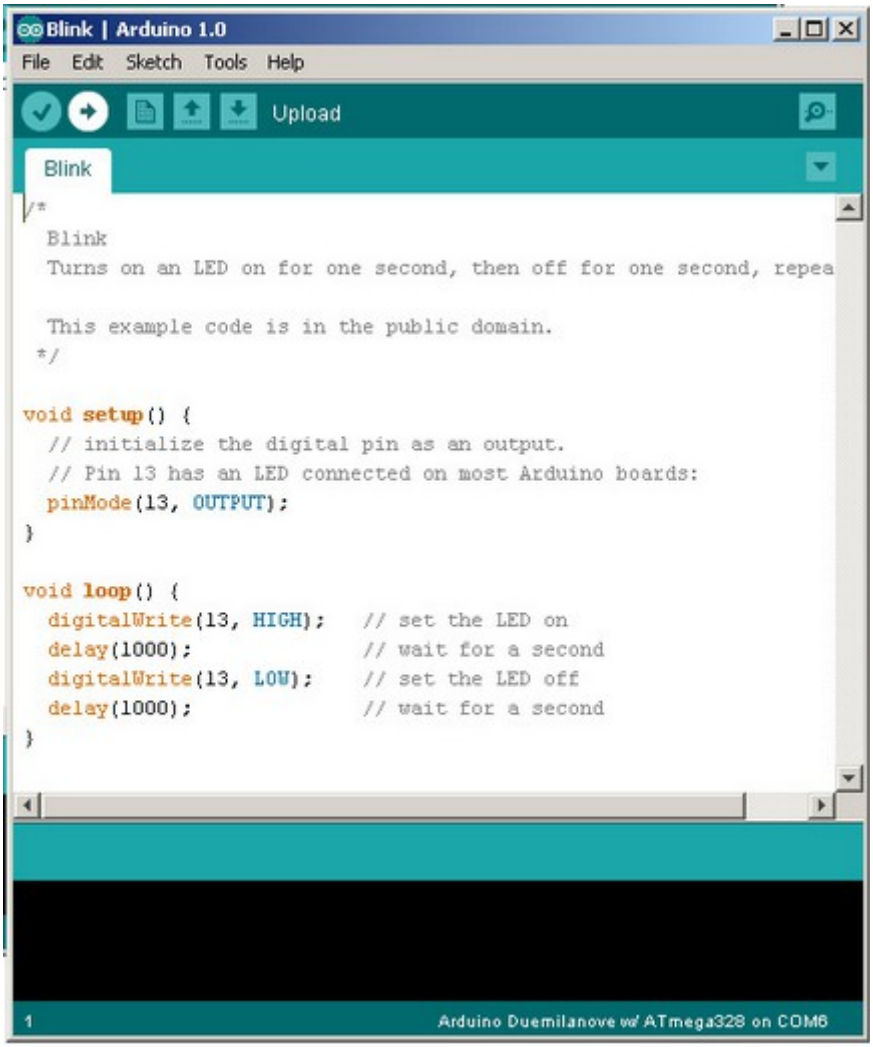
Autres composants

Nous avons aussi employé des transistors, des circuits intégrés, des moteurs, des photorésistances, des servomoteurs ; il existe d'autres composants, comme les condensateurs ou les bobines, mais nous n'avons pas eu l'occasion de les utiliser.

Annexe 6 : l'environnement de développement intégré C++

La programmation d'une carte Arduino est facilitée par l'existence d'un environnement de développement intégré (EDI, ou IDE en anglais, pour *integrated development environment*) dont le langage de programmation est C++.

Une fenêtre de cet environnement ressemble à :



Copie d'écran de la fenêtre de l'EDI C++

Le programme C++ est d'abord compilé (icône ), puis, si aucune erreur de compilation n'a été détectée, est téléchargé sur l'Arduino via le câble USB (icône ).

Écrire un programme Arduino consiste à définir deux fonctions :

- fonction `void setup()`, qui sert à initialiser l'Arduino, qui sera exécutée une fois juste après le lancement du programme
- fonction `void loop()`, qui sera exécutée ad vitam æternam, une fois la fonction `setup` terminée.

Comme le montrent les différents exemples du rapport, les principales fonctions sont :

- `pinMode`, qui permet de définir si les ports numériques seront programmés en entrée ou en sortie
- `digitalRead`, qui permet de lire l'état d'un port numérique programmé en entrée
- `digitalWrite`, qui permet de modifier l'état d'un port numérique programmé en sortie
- `analogRead`, qui permet de lire la valeur numérique (0 à 1023) de la tension appliquée sur un port

analogique

- `analogWrite`, qui permet de générer un signal MLI (PWM) sur un port numérique programmé en sortie et autorisant ce mode, à partir d'une valeur numérique (0 à 255).