

Cours d'initiation à Arduino

Sommaire

Introduction.....	2
I) Présentation de la carte	4
A) Qu'est ce qu'un microcontrôleur.....	4
B) Caractéristiques techniques de l'Arduino UNO	4
C) Un atout : les shields.....	5
II) Présentation du logiciel.....	6
A) IDE Arduino	6
B) Langage Arduino	7
III) Fonctionnalité de base	12
A) Les entrées/sorties.....	12
B) La gestion du temps	14
C) Les Interruptions	15
IV) Pratique	17
A) Hello LED	17
B) Push the button	17
C) Turn the potentiometer	17
D) Corrections.....	18
V) Quelques librairies.....	22
Conclusion	24

Introduction

Ce cours à pour but de présenter et d'initier à l'utilisation d'Arduino. Les cartes Arduino sont conçues pour réaliser des prototypes et des maquettes de cartes électroniques pour l'informatique embarquée.

Ces cartes permettent un accès simple et peu coûteux à l'informatique embarquée. De plus, elles sont entièrement libres de droit, autant sur l'aspect du code source (Open Source) que sur l'aspect matériel (Open Hardware). Ainsi, il est possible de refaire sa propre carte Arduino dans le but de l'améliorer ou d'enlever des fonctionnalités inutiles au projet.

Le langage Arduino se distingue des langages utilisés dans l'industrie de l'informatique embarquée de par sa simplicité. En effet, beaucoup de bibliothèques et de fonctionnalités de base occulte certains aspects de la programmation de logiciel embarquée afin de gagner en simplicité. Cela en fait un langage parfait pour réaliser des prototypes ou des petites applications dans le cadre de hobby.

Les possibilités des cartes Arduino sont énormes, un grand nombre d'applications ont déjà été réalisées et testées par bon nombre d'internautes. On retrouve par exemple diverse forme de robot (voir Fig.1), des stations météo (voir Fig.2). D'autres exemples d'applications sont disponibles sur le web aux adresses suivantes :

<http://www.hackaday.com/category/arduino-hacks/>

<http://www.instructables.com/tag/type-id/category-technology/channel-arduino/>

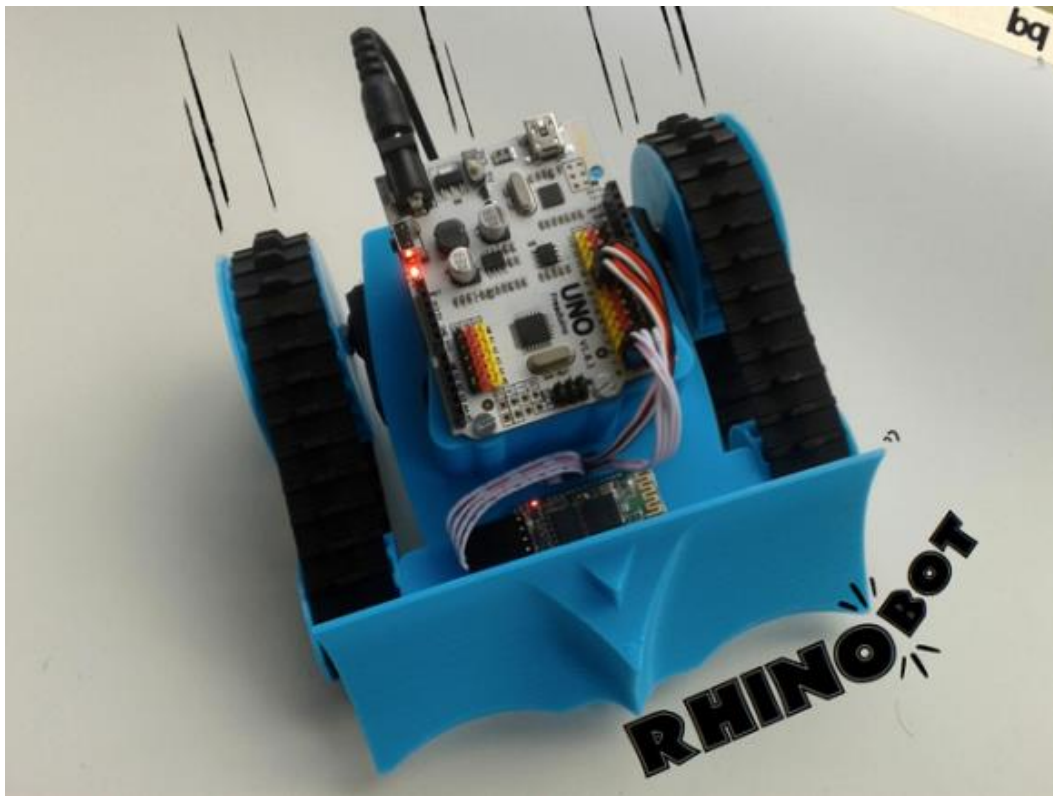


Figure 1 - Robot Arduino

<http://hackaday.com/2014/04/28/opensource-rhinobot-is-well-suited-for-hacking-and-sumo-robotics/>



Figure 2 - Station météo

<http://www.instructables.com/id/Google-Weather-on-graphical-display-with-Arduino/>

I) Présentation de la carte

A) Qu'est ce qu'un microcontrôleur

Les cartes Arduino font partie de la famille des **microcontrôleurs**. Un microcontrôleur est une petite unité de calcul accompagné de mémoire, de ports d'entrée/sortie et de périphériques permettant d'interagir avec son environnement. Parmi les périphériques, on recense généralement des Timers, des convertisseurs analogique-numérique, des liaisons Séries, etc. On peut comparer un micro contrôleur à un ordinateur classique, mais système d'exploitation et avec une puissance de calcul considérablement plus faible.

Les microcontrôleurs sont inévitables dans les domaines de l'informatique embarquée, de l'automatique et de l'informatique industrielle. Ils permettent de réduire le nombre de composant et de simplifier la création de cartes électroniques logiques.

B) Caractéristiques techniques de l'Arduino UNO

Un des modèles les plus répandu de carte Arduino est l'Arduino UNO (voir Fig.3). C'est la première version stable de carte Arduino. Elle possède toutes les fonctionnalités d'un microcontrôleur classique en plus de sa simplicité d'utilisation.

Elle utilise une puce ATmega328P [1] cadencée à **16Mhz**. Elle possède **32ko** de **mémoire flash** destinée à recevoir le programme, **2ko** de **SRAM** (mémoire vive) et **1 ko d'EEPROM** (mémoire morte destinée aux données). Elle offre **14 pins** (broches) d'entrée/sortie numérique (données acceptée 0 ou 1) [2] dont **6** pouvant générer des **PWM** (Pulse Width Modulation, détaillé plus tard). Elle permet aussi de mesurer des grandeurs analogiques grâce à ces **6 entrées analogiques** [3]. Chaque broche est capable de délivrer un courant de **40mA** pour une tension de **5V**.

Cette carte Arduino peut aussi s'alimenter et communiquer avec un ordinateur grâce à son port USB [4]. On peut aussi l'alimenter avec une alimentation comprise en 7V et 12V grâce à son connecteur *Power Jack* [5].

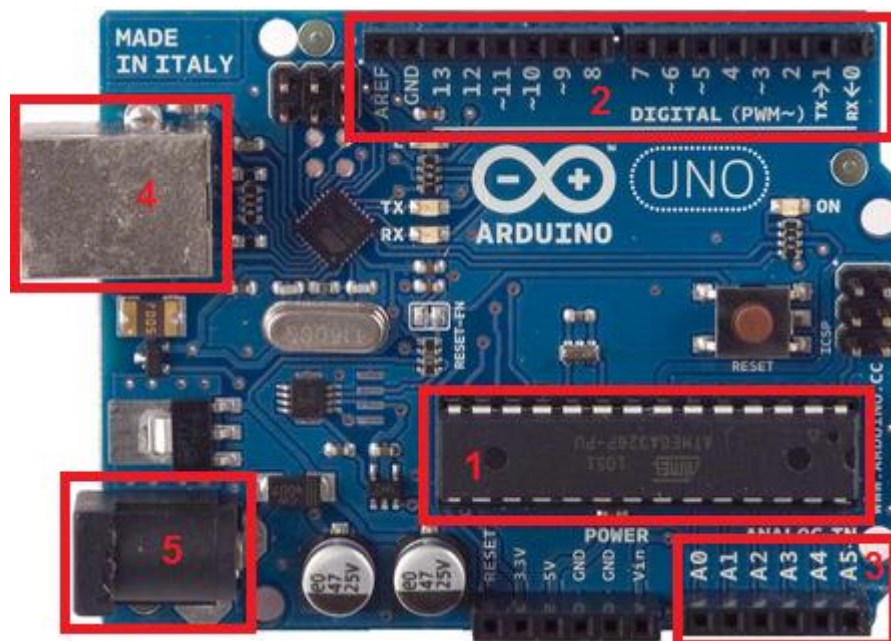


Figure 3 - Arduino UNO

C) Un atout : les shields

Pour la plupart des projets, il est souvent nécessaire d'ajouter des **fonctionnalités** aux cartes Arduino. Plutôt que d'ajouter soit même des composants extérieurs (sur une platine d'essai, circuit imprimé, etc.), il est possible d'ajouter des **shields**. Un shield est une carte que l'on connecte directement sur la carte Arduino qui a pour but d'ajouter des composants sur la carte. Ces *shield* viennent généralement avec une **librairie** permettant de les contrôler. On retrouve par exemple, des *shields* Ethernet, de contrôle de moteur, lecteur de carte SD, etc.

Le principal avantage de ces *shields* est leurs simplicités d'utilisation. Il suffit des les emboîter sur la carte Arduino pour les connecter, les circuits électronique et les logiciels sont déjà faits et on peut en empiler plusieurs. C'est un atout majeur pour ces cartes pour pouvoir tester facilement de nouvelles fonctionnalités. Cependant il faut bien garder à l'esprit que les *shields* ont un prix. Suivant les composants qu'ils apportent, leurs prix peuvent aller de 5 à 100€ !

II) Présentation du logiciel

A) IDE Arduino

Un **IDE** (environnement de développement) libre et gratuit est distribué sur le site d'Arduino (compatible Windows, Linux et Mac) à l'adresse <http://arduino.cc/en/main/software>. D'autres alternatives existent pour développer pour Arduino (extensions pour CodeBlocks, Visual Studio, Eclipse, XCode, etc.) mais nous n'aborderons dans ce cours que l'IDE officiel.

L'interface de l'IDE Arduino est plutôt simple (voir Fig.4), il offre une interface minimale et épurée pour développer un programme sur les cartes Arduino. Il est doté d'un éditeur de code avec **coloration syntaxique** [1] et d'une **barre d'outils rapide** [2]. Ce sont les deux éléments les plus importants de l'interface, c'est ceux que l'on utilise le plus souvent. On retrouve aussi une barre de menus [3] plus classique qui est utilisé pour accéder aux fonctions avancées de l'IDE. Enfin, une **console** [4] affichant les résultats de la compilation du code source, des opérations sur la carte, etc.

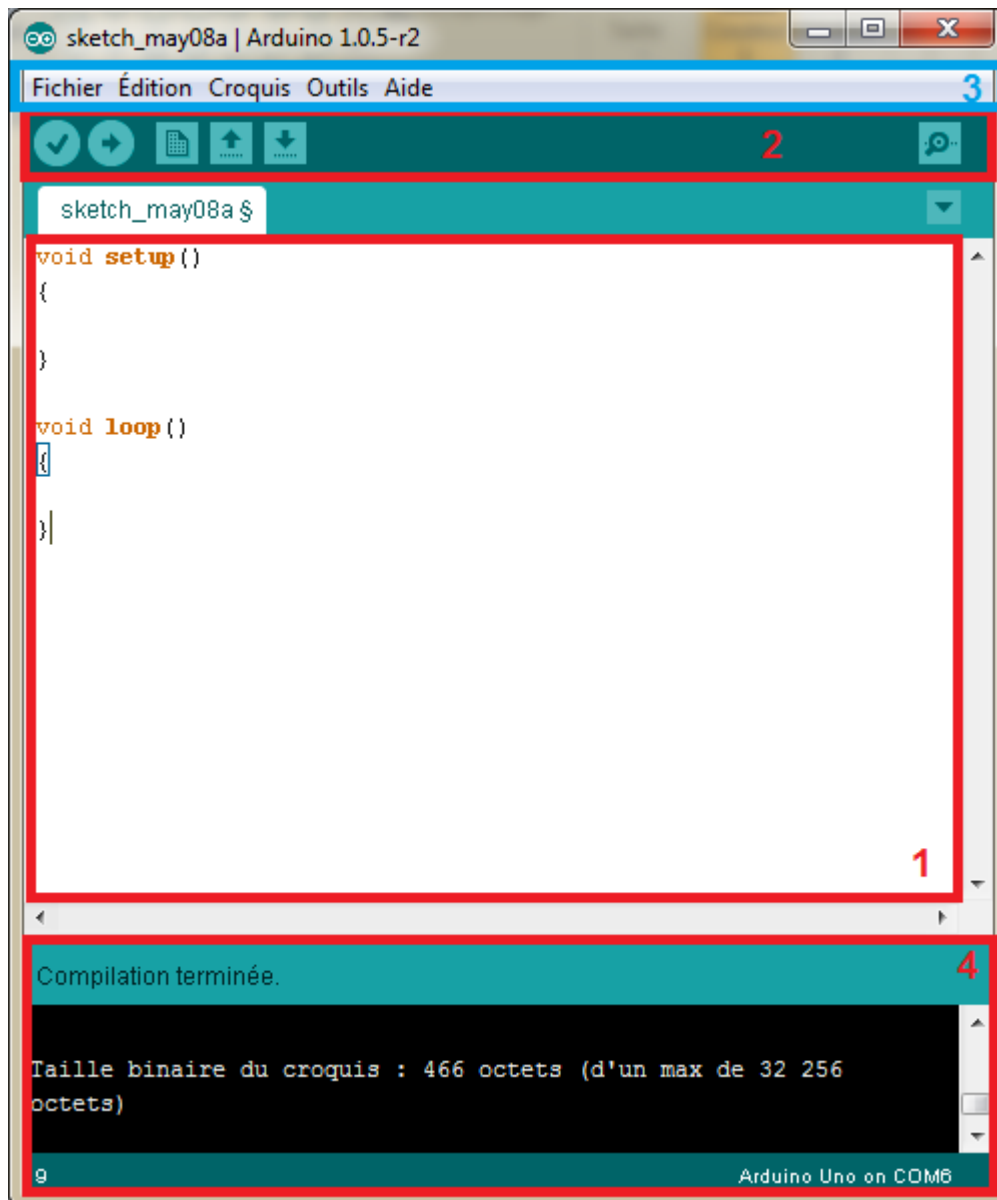


Figure 4 - IDE Arduino

B) Langage Arduino

Le langage Arduino est inspiré de plusieurs langages. On retrouve notamment des similarités avec le C, le C++, le Java et le Processing. Le langage impose une structure particulière typique de l'informatique embarquée. La fonction **setup** contiendra toutes les opérations nécessaires à la configuration de la carte (directions des entrées sorties, débits de communications série, etc.). La fonction **loop** elle, est exécutée en boucle après l'exécution de la fonction **setup**. Elle continuera de boucler tant que la carte n'est pas mise hors tension,

redémarrée (par le bouton **reset**). Cette boucle est absolument nécessaire sur les microcontrôleurs étant donné qu'il n'y a pas de système d'exploitation. En effet, si l'on omettait cette boucle, à la fin du code produit, il sera impossible de reprendre la main sur la carte Arduino qui exécuterait alors du code aléatoire.

Au niveau de la **syntaxe**, on retrouve des similarités avec les langages précédemment cités. La déclaration des variables se fait généralement dans l'espace global (de façon à partager les variables les plus importantes entre les deux fonctions principales). On retrouve les types de base suivant :

Nom	Contenu	Taille (en octet)	Plage de valeurs
<i>(unsigned) char</i>	Entier ou caractère	1	(0->255) -128 -> 127
<i>(unsigned) int</i>	Entier	2	(0->65 535) -32 768 -> 32 767
<i>(unsigned) long</i>	Entier	4	(0 -> 4 294 967 295) -2 147 483 648 -> 2 147 483 647
<i>float/double</i>	Nombre à virgule flottante	4	-3,4028235E+38 -> 3,4028235E+38
<i>String</i>	Chaine de caractères (Objet)	variable	Aucune
<i>boolean</i>	Booléen	1	True / False

Il existe d'autres types de base mais ils ne sont qu'un alias de ceux cités précédemment, la liste des types est disponible sur la page des références du site Arduino (<http://arduino.cc/en/Reference/HomePage>). La déclaration des variables suit cette syntaxe:

(const) <type> <nom>([<longueur du tableau>]) (= valeur);

Exemples :

const int constante = 12 ;

float univers = 42.0 ;

char lettre = 'b' ;

String chaine = "Hello World " ;

long tableau[12] ;

boolean vrai = *true* ;

On retrouve les **opérateurs** les plus courants pour les types de bases. Parmi eux, = (affectation), == (comparaison), != (différence), <, >, <=, >=, && (et logique), || (ou logique), ! (non logique). On retrouve aussi les opérateurs mathématiques (+, -, *, /, %) et les opérateurs logiques bit à bit (^ (XOR), & (et), | (ou), ~(non), << (décalage logique à gauche), >> (décalage logique à droite)).

Les **structures de contrôle** sont elles aussi similaires aux langages de références. On y retrouve toutes les structures de contrôle standard, conditions, boucle, switch, fonctions, etc. On peut aussi écrire des structures et des classes. Chaque structure de contrôle est suivie d'un bloc d'instructions délimitées par des accolades. Voici une liste des structures de contrôles les plus utilisées :

Nom	Utilité	Syntaxe
If-else	Condition logique	<pre><i>If</i> (<valeur booléenne>) { <instruction> } <i>else</i> { <instruction> }</pre>

If-else if - else	Condition logique multiples	<pre> <i>If</i> (<valeur booléenne>) { <instruction> } <i>else if</i> (<valeur booléenne>) { <instruction> } <i>else</i> { <instruction> } </pre>
Switch	Sélecteur	<pre> <i>Switch</i> (<variable>) { <i>case</i> <valeur> : <instruction> <i>break</i> ; <i>default</i> : <instruction> } </pre>
While	Boucle	<pre> <i>While</i> (<valeur booléenne>) { <instruction> } </pre>
For	Boucle itérative	<pre> <i>For</i> (<initialisation> ; <valeur booléenne> ; <évolution>) { <instruction> } </pre>

Voici quelques exemples d'utilisation de structure de contrôle.

```

If ( variable < 2 ){
  switch (variable) {
    case 1 :
      doSomething(10);
    break ;
    case 0 :
      doSomething(1);
    default :

```

```
        doSomething(0);  
    }  
} else if ( variable > 4 ) {  
    while ( variable != 10 ) {  
        variable = doSomethingElse(variable-1);  
    }  
} else {  
    for ( int i = 0 ; i < 10 ; i++ ) {  
        variable = variable *2 ;  
    }  
}
```

Les **fonctions**, les **structures** et les **classes** se déclarent de la même façon qu'en **C++**.
Elles se déclarent sous cette forme :

Structure :

```
struct <nom> {  
    <type> <nom du champ> ;  
};
```

Fonction :

```
<type de retour> <nom>(<paramètre>) {  
    <instruction>  
}
```

Classe :

```
class <nom> {  
    public :  
    <attributs et champ publics>  
    private :  
    <attributs et champ privés>  
    protected :
```

<attribut et champ privés>

};

III) Fonctionnalité de base

A) Les entrées/sorties

Le langage Arduino vient avec un nombre important de **fonction de base** permettant d'interagir avec son **environnement**. Les fonctions les plus utilisées sont les fonctions d'entrée/sorties. Ce sont elles qui permettent **d'envoyer** ou de **mesurer** une tension sur une des broches de la carte.

Dans un premier temps, avant d'effectuer une mesure ou d'envoyer une commande. Il est nécessaire de définir la **direction** des broches utilisées. Pour cela on fait appel à la fonction *pinMode* en lui donnant d'une part, la broche concernée, et d'autre part, la direction :

```
void setup() {  
    pinMode(1,OUTPUT); // Broche 1 en sortie  
    pinMode(2,INPUT);  // Broche 2 en entrée  
}
```

Une fois cette configuration faite, on peut procéder à l'utilisation des broches. Toutes les broches sont capables **d'écrire** et de **lire** des données **numériques** (c'est-à-dire des 0 (0V) ou des 1 (5V)). Mais, certaines disposent de fonctionnalité supplémentaire.

Tout d'abord, toutes les cartes Arduino possèdent des **entrées analogiques**. Ce sont les broches **A0-A1-A2 etc.** Elles permettent de lire des tensions analogiques (comprise entre **0 et 5V**) et de le convertir en entier (compris entre **0 et 1023**) **proportionnellement** à la tension mesurée. Certaines cartes Arduino possèdent des **sorties analogique** faisant l'opération inverse (met une tension sur la broche proportionnellement à l'entier donné), mais ce n'est pas le cas pour l'Arduino UNO.

Pour pouvoir tout de même contrôler des composants autrement qu'en « *tout ou rien* » il est possible d'utiliser des broches **PWM**. Ce sont les broches annotés par un tilde ~ sur la carte. Les PWM (Pulse Width Modulation) sont utilisées pour synthétiser des signaux analogiques en modulant le temps passé à l'état 1 (5V). Le signal obtenu est représenté figure 5. En utilisant une fréquence relativement élevée, les PWM permettent de commander certains composants **comme si** il recevait une **tension analogique**. Cela provient du fait que les composants utilisés dans l'électronique analogique, ne changes pas **d'états instantanément**. Par exemple, une ampoule à incandescence reste chaude et éclaire un court instant après avoir été éteinte. Ce phénomène est généralement invisible à l'œil nu. Grâce à elles, on pourra par exemple faire varier l'intensité d'une LED. La plupart des cartes Arduino utilisent des PWM cadencées à **490Hz** environ.

Toutes ces fonctionnalités sur les broches d'entrées sorties sont utilisables par le biais de quatre fonctions :

- ***digitalRead(pin)*** : mesure une donnée numérique sur une des broches, la broche en question doit être réglée en entrée.
- ***digitalWrite(pin, value)*** : écrit une donnée numérique sur une des broches, la broche concernée doit être réglée en sortie. Le paramètre *value* doit être égal à *HIGH* (état 1 soit 5V) ou *LOW* (état 0 soit 0V).
- ***analogRead(pin)*** : mesure une donnée analogique sur une des broches (compatible seulement), la broche doit être réglée sur entrée.
- ***analogWrite(pin, value)*** : écrit une donnée sous forme de PWM sur une des broches (compatible uniquement), la broche doit être réglée en sortie. Le paramètre *value* doit être compris dans l'intervalle *[0;255]*

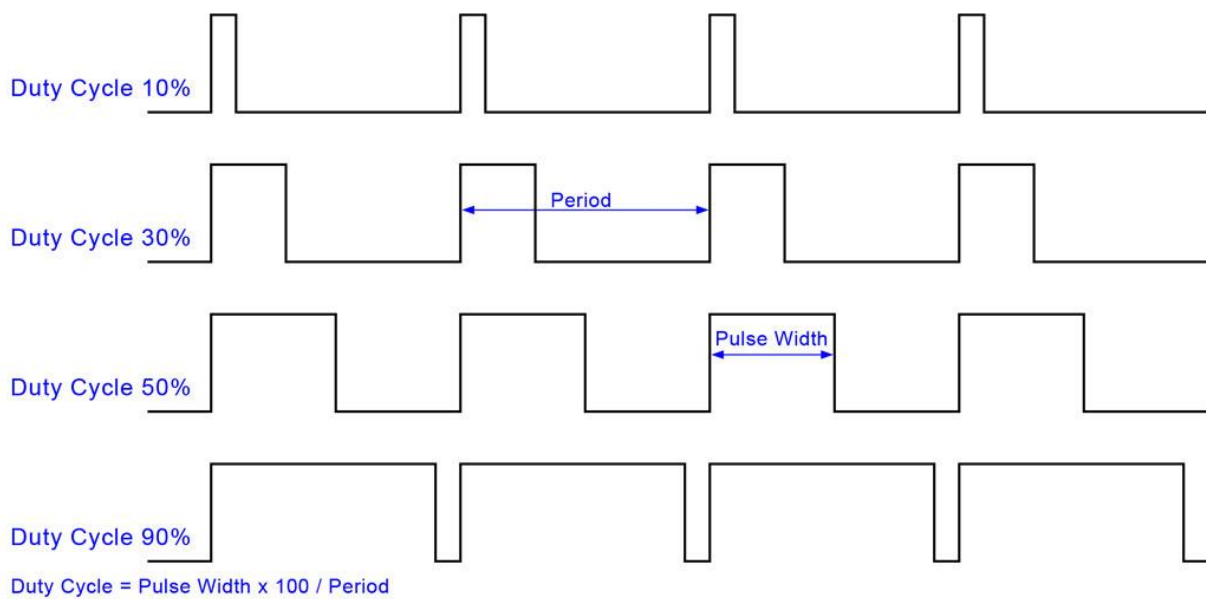


Figure 5 - Signal PWM

B) La gestion du temps

Pour la plupart des applications de **domotique**, il est nécessaire de faire intervenir des **intervalles de temps**. Par exemple, pour gérer le temps d'appui sur un bouton ou pour faire une sonnerie qui se répète un certains nombre de fois. Le langage Arduino fournis quelques fonctions permettant de gérer le temps.

Il est possible d'insérer une **pause** dans son programme pendant un instant. Pour cela, on utilise les fonctions **delay** et **delayMicroseconds** qui insère une pause suivant le paramètre passé (en milliseconde pour l'un, en microseconde pour l'autre). Cependant ces fonctions **bloquent** le microcontrôleur, on ne peut alors plus effectuer aucune action.

En plus d'insérer une pause, il est possible de **mesurer le temps**. De la même manière que les fonctions de délai, on utilise les fonctions **millis** et **micros** qui donnent le nombre de **milliseconde** (respectivement **microseconde**) depuis le lancement de la carte. Attention, ces fonctions incrémente une variable (interne). Ces variables **se remettent à zéro** une fois le maximum atteint (**overflow**). La variable utilisée pour les millisecondes atteint sont maximum au bout de 49 jours et 17 heures et la variable utilisée pour les microsecondes au bout de 71 minutes et 34 secondes environ. Il faut donc faire attention lors de l'utilisation de ces fonctions pour des utilisations longues durées.

C) Les Interruptions

Il est parfois nécessaire en informatique embarquée, d'attendre un événement externe (appui sur un bouton, données d'un capteur, etc.) pour effectuer une action. Pour ce type de problème, on utilise les **interruptions**. Les interruptions sont des **portions de code** (fonctions) appelés lorsque qu'un **événement** (interne ou externe) survient et à besoin d'être traité sur le champ. Il faut cependant faire attention, ce mécanisme **interrompt** le code exécuté, il est **prioritaire** par rapport au reste du code. Vu qu'il est possible de mesurer les événements ponctuellement (via les fonctions d'entrées/sorties) on utilise généralement les interruptions pour du **code critique** (arrêt d'urgence par exemple) ou des événements **non-ponctuels** (transmissions de données depuis un ordinateur par exemple).

Aussi, le nombre d'interruption externe est **limité sur à 2** sur la plupart des cartes Arduino. Les interruptions sont utilisables sur les **broches compatibles** seulement (broches 2 et 3 sur l'Arduino UNO). Pour choisir la fonction et la broche utilisée pour l'interruption, on utilise la fonction ***attachInterrupt***. On peut utiliser ***detachInterrupt*** pour supprimer l'interruption. Il est possible de partir en interruptions sur 4 types d'événements :

- **LOW** : Lorsque la broche est à l'état 0 (0V)
- **RISING** : Lorsque la broche passe de l'état 0 (0V) à l'état 1 (5V) (front montant).
- **FALLING** : Lorsque la broche passe de l'état 1 (5V) à l'état 0 (0V) (front descendant).
- **CHANGE** : Lorsque la broche change d'état (front montant et front descendant).

Voici un exemple d'utilisation :


```
volatile boolean etat = false ;  
void appuiBouton() {  
    etat = !etat ; // Changement d'état  
}  
void setup() {  
    pinMode(2, INPUT) ; // Broche 2 en entrée  
    attachInterrupt(0, appuiBouton, RISING) ; // On attache à l'interruption 0 (broche 2) la  
                                                fonction appuiBouton sur un front montant  
}
```

On remarque l'apparition du mot clef **volatile** avant la déclaration de la variable **etat**. Ce mot clef est nécessaire pour toutes les variables qui sont modifiées dans une interruption. Cela a une incidence sur la manière dont le compilateur traite l'accès à la variable.

Il est parfois nécessaire de **désactiver temporairement** les interruptions par exemple lorsque l'on exécute du code critique (activation d'un moteur, etc.). Deux fonctions permettent de changer l'activation des interruptions **interrupts** et **noInterrupts** pour activer (respectivement désactiver) les interruptions.

IV) Pratique

A) Hello LED

Nous allons maintenant passer à la pratique. Pour cette première manipulation, il est demandé de faire clignoter une LED de façon régulière. Vous pouvez soit utiliser une LED avec une résistance en série sur une *breadboard*. Ou alors, vous pouvez utiliser la LED directement sur l'Arduino (broche 13 sur la carte Arduino UNO).

Si vous souhaitez plus de difficulté, essayer de faire ce même programme en utilisant la fonction *millis* plutôt que *delay*.

B) Push the button

Maintenant, nous allons expérimenter la mesure sur une des broches. Le but est d'allumer une LED selon si un bouton est enfoncé ou non. Pour le câblage du bouton, il est nécessaire d'y mettre en série une résistance relié à la masse. Cela à pour but d'éviter les court circuits lorsque le bouton est enfoncé.

Ici aussi, on peut augmenter la difficulté en utilisant les interruptions plutôt que la lecture directe de la broche.

C) Turn the potentiometer

Pour terminer, nous allons nous intéresser aux fonctions « analogiques ». Il est maintenant question d'allumer une LED et de faire varier son intensité en fonction de la position d'un potentiomètre.

Pour augmenter la difficulté, on pourrait ajouter quelques LEDs et déterminer la LED allumée à allumer en fonction de la valeur du potentiomètre en plus de faire varier leurs intensités.

D) Corrections

Hello LED

Code avec *delay* :

```
const int pinLed = 13;

void setup()
{
    pinMode(pinLed, OUTPUT); // Broche 13 en sortie
}

void loop()
{
    delay(500); // Attente d'une demi seconde
    digitalWrite(pinLed, HIGH); // Allumage de la LED
    delay(500);
    digitalWrite(pinLed, LOW); // Eteignage de la LED
}
```

Code avec *millis* :

```
const int pinLed = 13;
int temps;
int etat;

void setup()
{
    pinMode(pinLed, OUTPUT); // Broche 13 en sortie
    etat = LOW; // LED éteinte
}

void loop()
{
    int present = millis();

    if ( temps+500 < present ) // Vérification du chronomètre
    {
        temps = present; // Actualisation du chronomètre
        etat = !etat; // Changement d'état
        digitalWrite(pinLed, etat); // Changement d'état de la LED
    }
}
```

Push the button

Code sans interruptions :

```
const int pinLed = 13;
const int pinBouton = 2;

void setup()
{
```

```

    pinMode(pinLed, OUTPUT); // Broche 13 en sortie
    pinMode(pinBouton, INPUT); // Broche 2 en entrée
}

void loop()
{
    int etat = digitalRead(pinBouton); // On lit la valeur du bouton
    digitalWrite(pinBouton, etat); // On allume (ou non) la LED

    delay(50); // On peut mettre un delay pour éviter les allumages
               // intempestifs
}

```

Code avec interruption :

```

const int pinLed = 13;
const int pinBouton = 2;
volatile int temps;
volatile boolean etat;

void interruption()
{
    int present = millis(); // Lecture du chronomètre
    // Attention, la valeur de millis() NE VARIE PAS
    // pendant une interruption ! On l'utilise ici
    // seulement pour éviter un changement d'état
    // intempestif
    if ( temps+100 < present ) // Vérification du chronomètre
    {
        etat = !etat; // Changement d'état
        temps = present;
    }
}

void setup()
{
    pinMode(pinLed, OUTPUT); // Broche 13 en sortie
    pinMode(pinBouton, INPUT); // Broche 2 en entrée
    attachInterrupt(0, interruption, CHANGE); // Mise en place de
l'interruption
}

void loop()
{
}

```

Turn the potentiometer

Code pour une LED :

```

const int pinLed = 11;
const int pinPotar = A0;

void setup()
{
    pinMode(pinLed, OUTPUT); // Broche 11 en sortie (broche PWM)
}

```

```
    pinMode(pinPotar, INPUT); // Broche A0 en entrée (broche analogique)
}

void loop()
{
    int valeur = analogRead(pinPotar); // Lecture de la valeur du
    potentiomètre

    valeur = map(valeur,0,1023,0,255); // Change l'intervalle de valeur
    [0;1023]->[0;255]

    analogWrite(pinLed,valeur); // Écriture sur la broche 11 en PWM
    delay(50);
}
```

Code pour plusieurs LEDs (3):

```
const int pinLed[3] = {9,10,11};
const int pinPotar = A0;
const int nbLed = 3;

void setup()
{
    for ( int i = 0 ; i < nbLed ; i++ )
    {
        pinMode(pinLed[i], OUTPUT); // Broche LED en sortie (broche PWM)
    }

    pinMode(pinPotar, INPUT); // Broche A0 en entrée (broche analogique)
}

void loop()
{
    int valeur = analogRead(pinPotar); // Lecture de la valeur du
    potentiomètre
    int ledCourante;

    valeur = map(valeur,0,1023,0,255); // Change l'intervalle de valeur
    [0;1023]->[0;255]

    if ( valeur < 85 ) // Choix de la LED
    {
        ledCourante = 0;
    }
    else if ( valeur < 170 )
    {
        ledCourante = 1;
    }
    else
    {
        ledCourante = 2;
    }

    for ( int i = 0 ; i < nbLed ; i++ ) // Allumage de la led donnée
    {
        if ( i == ledCourante )
        {
            // Allumage de la LED correspondante en fonction de la tension

```

```
        // lue sur le potentiomètre
        analogWrite(pinLed[i], (valeur%(85))*3);
    }
    else
    {
        analogWrite(pinLed[i], 0);
    }
}

delay(50);
}
```

V) Quelques librairies

En plus de la simplicité du langage et des nombreuses fonctionnalités qu'offre. L'IDE vient avec un nombre important de **librairies** évitant ainsi d'implémenter des fonctions courantes dans l'informatique embarquée.

Une des librairies les plus utilisée est celle implémentant la **communication série**. La majorité des cartes Arduino possède un émulateur de connexion série pour communiquer au travers de l'USB. Ainsi, on peut **communiquer** avec l'ordinateur sur lequel la carte Arduino est connectée. Cela permet, par exemple, de déboguer un programme en **affichant la valeur des variables** ou simplement afficher la valeur des capteurs. Cette librairie a été directement implémentée dans le langage Arduino. On peut accéder à la communication série (au travers l'USB) grâce à l'objet **Serial**. Une autre librairie existe pour communiquer par liaison série via une des broches de la carte.

Il est parfois nécessaire de **stocker des informations** même après l'arrêt de la carte. Il est possible de stocker une petite quantité d'information sur la **mémoire EEPROM** (*Electrically Erasable Programmable Read Only Memory*) intégrée. Une librairie est aussi fournie pour dialoguer avec cette mémoire. Il est possible de **lire** et **d'écrire** sur cette mémoire **sans rien ajouter** sur la carte. Cette mémoire est accessible via l'objet **EEPROM** et en ajoutant la librairie du même nom. Attention, la mémoire EEPROM a une **durée de vie limitée** (environ 100 000 cycles d'écritures). Il faut donc veiller à ne pas écrire **répétitivement** les données. D'autres alternatives existent pour ajouter de plus grandes quantités de mémoires mortes à l'Arduino (carte SD notamment) mais cela demande l'ajout de composants externes.

Pour terminer, il existe aussi des librairies permettant de **contrôler des composants externes**. Parmi ces librairies, une des plus populaires est celle contrôlant des **servomoteurs**. Un servomoteur est un composant électronique composé d'un **moteur** et d'une **carte d'asservissement**. On peut le contrôler en position (c'est-à-dire choisir l'angle du moteur) grâce aux **PWM**. Une librairie Arduino est implémentée pour **contrôler simplement** ces moteurs : la librairie **Servo**. Il faut toutes fois faire attention à éviter d'alimenter les

servomoteurs **directement** sur une des broches d'alimentation de la carte. En effet, un servomoteur consomme une **quantité important de courant** (suivant le couple exercé par le moteur). Il est donc fortement conseillé de relier les servomoteurs sur des alimentations **externes** (batteries, piles de 9V, etc.).

Conclusion

On peut conclure sur le fait que les cartes Arduino sont un puissant outil de **prototypage** pour les cartes électroniques. Mais aussi, elles permettent un **accès facile et intuitif** à l'informatique embarqué. On pourra ainsi enrichir tout ces projets d'un **microcontrôleur** pour leurs donner une plus value importante.

L'Arduino UNO est une des cartes les plus courantes. C'est la première de ce genre. Il existe cependant **d'autres versions** de cartes Arduino plus adaptées pour certains projets. Pour certains projet il va falloir, par exemple, plus d'entrées/sorties. On pourra alors opter pour l'Arduino Méga.

Il ne faut cependant pas oublier qu'Arduino n'est pas **la seule marque** à proposer ce type de cartes. En effet, des concurrents on rapidement vu le jour. Il existe deux types de concurrences. Les cartes **compatibles Arduino**, ce sont des produits différents (de par les composants qu'ils proposent ou leurs prix) mais **entièrement compatible** avec le logiciel et les librairies Arduino (on peut par exemple citer *Freeduino*). Il existe aussi des cartes **similaires** mais qui ne sont pas compatible avec Arduino (par exemple le *LaunchPad* de *Texas Instrument*).

Pour plus d'informations sur le projet Arduino, de l'aide pour la réalisation d'un projet ou des documentations, reportez vous sur le site officiel Arduino : <http://www.arduino.cc/>.

Contact : lechalupe.julien@gmail.com

