Chapitre 4

Initiation à l'utilisation des cartes arduino

Objectifs :

- ✓ **Connaitre les différents types de cartes arduino.**
- ✓ **Connaitre les caractéristiques de la carte arduino UNO.**
- ✓ **Apprendre les bases de la programmation arduino.**

1. Introduction

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 couteux 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 à tout utilisateur de refaire sa propre carte Arduino dans le but de l'améliorer ou d'enlever des fonctionnalités inutiles à son projet.
Le langage Arduino se distingue des langages utilisés dans l'industrie de l'informatique embarquée par sa simplicité. En effet, beaucoup de librairies et de fonctionnalités de base sont disponibles ce qui permet de faciliter la réalisation des prototypes ou des petites applications.

2. Présentation de la carte arduino

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 trouve généralement des Timers, des convertisseurs analogique-numérique, des liaisons Séries, etc.
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 composants et de simplifier la création de cartes électroniques dédiées à la commande de processus industriels.

2.1 Différents types

Il existe plusieurs types de cartes arduino qui se distingue l'une de l'autre principalement par le type du microcontrôleur utilisé, le nombre d'entrées/sorties, la capacité de la mémoire et la vitesse de fonctionnement. On peut citer à titre indicatif l'arduino UNO, Mega, Leonardo, Due, Pro mini, micro,…
Les cartes arduino les plus répandues pour les applications courantes sont la carte UNO et la carte Mega dont les principales caractéristiques sont données au tableau suivant :

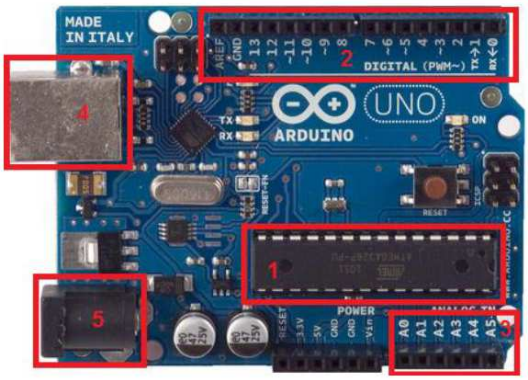
Carte	µC	E/S	Analog	PWM	SRAM	EPROM	Speed	Int	Série	SPI	I^2C
UNO	ATmega328	14	6	6	2K	1K	16MHz	2	1	1	1
Mega 2560	ATmega2560	54	16	14	8K	4K	16MHz	6	4	1	1

Dans la suite de ce cours on s'intéressera uniquement à la carte arduino UNO.

2.2 Caractéristiques techniques de l'Arduino UNO

L'Arduino UNO 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 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 numériques (données acceptée 0 ou 1) dont 6 pouvant générer des PWM (Pulse Width Modulation,). Elle permet assi de mesurer des grandeurs analogiques grâce à ces 6 entrées analogiques. Chaque broche est capable de délivré 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. On peut aussi l'alimenter avec unes alimentions comprise en 7V et 12V via son connecteur Jack.

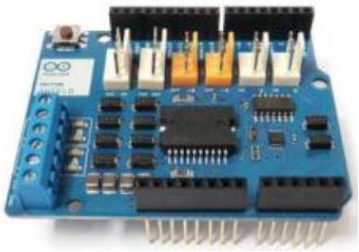


1. Microcontrôleur.
2. Entrées/Sorties numériques.
   (~ : sortie PWM).
3. Entrées analogiques.
4. Port USB.
5. Connecteur Power Jack.

2.3 Les shields

Un shield est une carte que l'on connecte directement sur la carte Arduino dans le but d'ajouter des composants supplémentaires sur la carte permettant ainsi d'assurer une fonction donnée. 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 emboiter sur la carte Arduino pour les connecter. Il est donc toujours possible de pour pouvoir tester facilement de nouvelles fonctionnalités.

Shield Ethernet

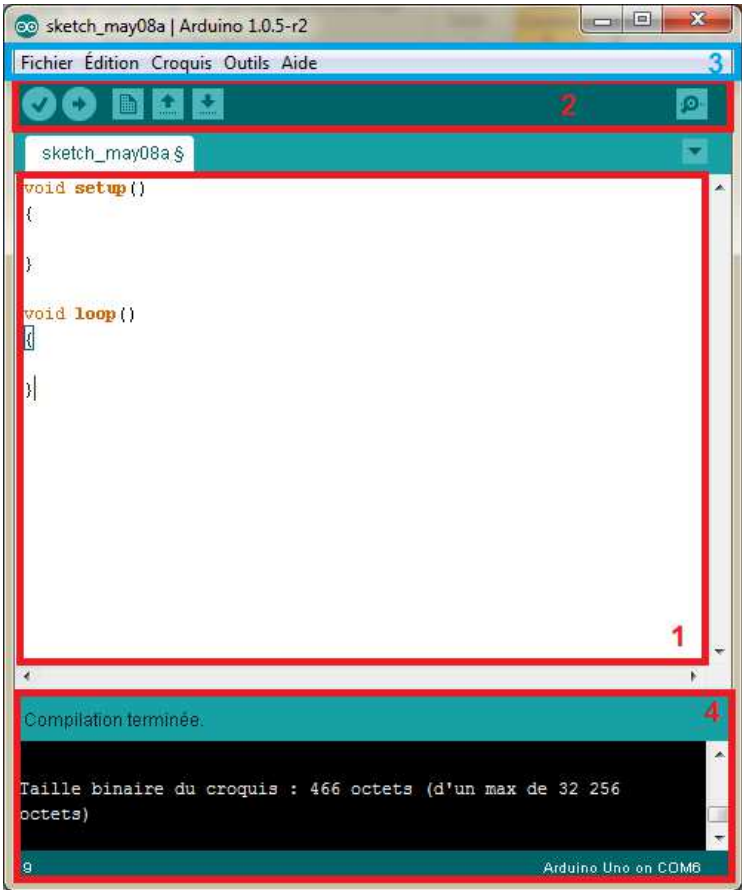


Shield de commande de moteurs

3. Présentation du logiciel

3.1 IDE Arduino

Un IDE (environnement de développement) libre et gratuit est distribué sur le site d'Arduino (compatible Windows, Linux et Mac). 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.



1. Editeur de code.
2. Barre d'outils rapide.
3. Barre de menus.
4. Console d'affichage.

3.2 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'on 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.

4. Fonctionnalités de base

4.1 Les entrées/sorties

Le langage Arduino vient avec un nombre important de fonctions 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 fonctionnalités supplémentaires.

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 analogiques 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.

Remarque

Les broches de la carte arduino UNO sont divisées en 3 groupes (ports) de la façon suivante :

- Port D : comprend les 8 pins de **0** à **7**.
- Port B : comprend les 6 pins de **8** à **13**.
- Port C : comprend les 6 pins de **A0** à **A5**.

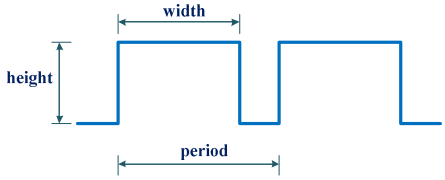
Pour définir la direction d'un port (entrée ou sortie) on utilise un registre de direction pour chaque port (DDRB, DDRC ou DDRD) :

DDRB = B111111 ;	permet de configurer les 6 pins du port B en sortie.
DDRB = B000000 ;	permet de configurer les 6 pins du port B en entrée.
DDRD = B00001111 ;	permet de configurer les 4 pins (0 à 3) du port D en sortie et les 4 pins (4 à 7) du même port en entrée.
DDRC = B110000 ;	permet de configurer les 4 pins (A0 à A3) en entrée et les 2 pins (A4 et A5) en sortie.

Après la configuration en sortie on peut envoyer les données vers les ports comme suite :

PORBT = B111111 ;	permet de mettre les 6 pins du port B à 1.
PORTB = B000000 ;	permet de mettre les 6 pins du port B à 0.
PORTD = B11110000 ;	permet de mettre les 4 pins (0 à 3) du port D à 0 et les 4 pins (4 à 7) du même port à 1.
PORTC = B000011 ;	permet de mettre les 4 pins (A0 à A3) à 1 et les 2 pins (A4 et A5) à 0.

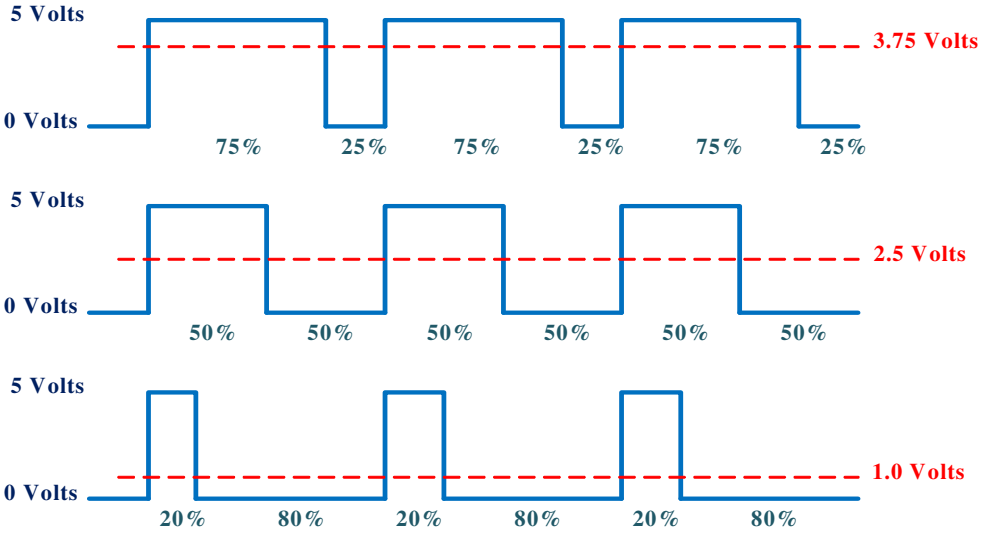
Pour pouvoir contrôler des composants autrement qu'en « tout ou rien » il est possible d'utiliser des broches PWM. Ce sont les broches annotés par un tilde ~ sur la carte. Les PWM (Pulse Width Modulation) sont utilisées pour synthétiser des signaux analogiques en modulant le temps passé à l'état 1 (5V).



La tension moyenne dépend alors du rapport cyclique du signal :

$$Tension\ moyenne\ de\ sortie = \frac{Width}{Period} \times Height$$

En faisant varier le rapport cycle c'est à dire le temps pendant lequel le signal est à 1, on fait varier la tension moyenne de sortie :



En utilisant une fréquence relativement élevée, les PWM permettent de commander certains composants comme si il recevait une tension analogique. On peut donc utiliser cette technique pour faire varier par exemple l'intensité d'une LED ou la vitesse d'un moteur à courant continu. La plupart des cartes Arduino utilisent des PWM cadencées à 490Hz environ.
Toutes ces fonctionnalités sur les broches d'entrées sorties sont utilisables au moyen 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 « A0 à A5»).
- ☒ 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]

4.2 Gestion du temps

Le langage Arduino fournis quelques fonctions permettant de gérer le temps. Il est possible donc d'insérer une pause dans le programme pendant une certaine durée 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 millisecondes (respectivement microsecondes) 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.

5. Applications

5.1 Clignotement d'une Led

❋ *Code avec delay*

```
//Clignotement d'une Led branchée sur la pin 13

//déclaration des variables
const byte Led=13;     //pin 13 pour la commande de la Led

//configuration des entrées/sorties
void setup() {
pinMode(Led,OUTPUT);
}
//programme principal
void loop() {
digitalWrite(Led,1);       //allumer la led
delay(1000);               //attente d'une seconde
digitalWrite(Led,0);       //éteindre la led
delay(1000);               //attente d'une seconde
}
```

❋ *Code avec millis*

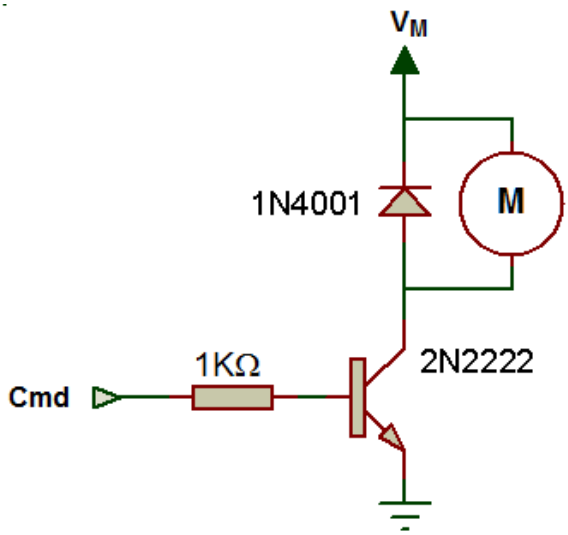
```
//Clignotement d'une Led branchée sur la pin 13

//déclaration des variables
const byte Led=13;     //pin 13 pour la commande de la Led
int temps;
int etat=0;

void setup() {
pinMode(Led,OUTPUT);
}
```

```
//programme principal
void loop() {
int present = millis();
if(temps + 1000 < present)
{
temps =  present;            //actualiser le chronomètre
etat = !etat ;               //complémenter la variable etat
digitalWrite(Led,etat);      //changement de l'état de la led
}
}
```

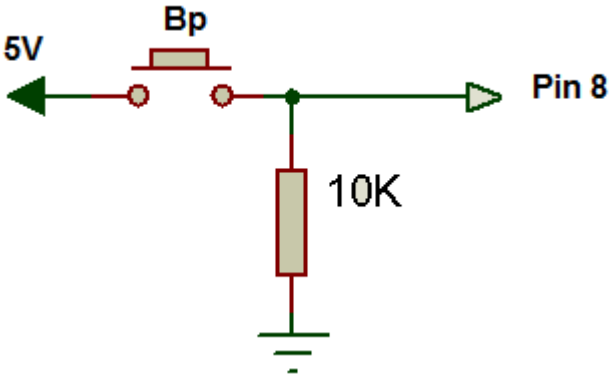
5.2 Pilotage d'un moteur à courant continu (DC)

5.2.1 Pilotage dans un seul sens à vitesse constante

Pour commander le moteur au moyen d'une carte arduino, on utilise un circuit intermédiaire (transistor ou autre) qui fourni la puissance nécessaire au moteur. Soit par exemple le circuit suivant :



On branche alors l'entrée **Cmd** du circuit ci-dessus à la pin 9 de la carte (par exemple) et un bouton poussoir à la pin 8 de la façon suivante :



Lorsque le bouton poussoir **Bp** est relâché (ouvert) la pin 8 se trouve à l'état 0 et s'il est appuyé (fermé) la pin 8 est reliée à 5V soit le niveau 1 logique.

☒ *Programme arduino*

```
//Commande d'un moteur à courant continu dans un seul sens de
//rotation et à vitesse constante

//déclaration des variables
const byte Moteur=9;     //pin 9 pour la commande du moteur
const byte Bouton=8;      //pin 8 pour lire l'état du bouton
byte etat=0;              //variable qui va recevoir l'état du bouton

//configuration des entrées/sorties
void setup() {
pinMode(Moteur,OUTPUT);
pinMode(Bouton,INPUT);
}

//programme principal
void loop() {
etat=digitalRead(Bouton); //lire l'état du bouton le socker dans la
//variable etat
if(etat==1)              //si le bouton est actionné
digitalWrite(Moteur,1);   //mettre en marche le moteur
else//sinon
digitalWrite(Moteur,0);   //arreter le moteur
}
```

Il est possible d'optimiser ce programme (programme principal) de la façon suivante :

```
//programme principal
void loop() {
etat=digitalRead(Bouton); //lire l'état du bouton le stocker dans la
//variable etat
digitalWrite(Moteur,etat); //envoyer au moteur l'état du bouton
}
```

5.2.2 Pilotage dans un seul sens à vitesse variable

Pour faire varier la vitesse on utilise une des sorties PWM de la carte arduino. Ces sorties sont marquées par le symbole ~ soient les pins 3, 5, 6, 9, 10 et 11.
Le pilotage s'effectue dans ce cas en envoyant, sur une sortie PWM, la commande suivante :

analogWrite(pin,vitesse)

où pin est une pin PWM et vitesse un entier compris entre 0 et 255.

Dans la suite nous allons choisir la pin 9 comme sortie PWM et nous allons choisir les valeurs :

- ✓ 50 pour faible vitesse,
- ✓ 120 pour vitesse moyenne
- ✓ 255 pour vitesse maximale.

☒ *Programme arduino*

Version1 : commande 3 vitesses

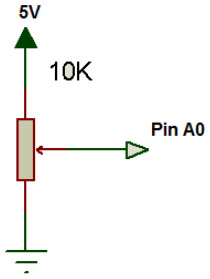
```
//Commande d'un moteur à courant continu dans un seul sens de
//rotation et à vitesse variable (3 vitesses : faible, moyenne et maximale)

//déclaration des variables
const byte Moteur=9;     //pin 9 pour la commande du moteur

//configuration des entrées/sorties
void setup() {
pinMode(Moteur,OUTPUT);
}
//programme principal
void loop() {
analogWrite(Moteur,50); //commande vitesse faible
delay(2000);//temporisation 2s
analogWrite(Moteur,120); //commande vitesse moyenne
delay(2000);//temporisation 2s
analogWrite(Moteur,255); //commande vitesse maximale
delay(2000);//temporisation 2s
analogWrite(Moteur,0); //arreter le moteur
delay(2000);//temporisation 2s
}
```

Version2 : lecture directe de la vitesse sur le moniteur série

```
//Commande d'un moteur à courant continu dans un seul sens de rotation
//et à vitesse variable. Lecture de la vitesse sur le moniteur série

//déclaration des variables
const byte Moteur=9; //pin 9 pour la commande du moteur
intvitesse;//variable pour calculer la vitesse

//configuration des entrées/sorties
void setup() {
pinMode(Moteur,OUTPUT);
Serial.begin(9600);  //ouvre le port serie ,fixele debit 9600 bauds
while(!Serial);    //tant qu'il n'y a pas de données
Serial.println("Iroduire la vitesse : entre 0 et 255"); //afficher ce message
}
//programme principal
void loop() {
if (Serial.available())          //s'il y a de données disponibles
{
vitesse = Serial.parseInt();  //lecture de la vitesse. La fonction parseInt()
                     //permet de convertir une chaine de chiffres en
   //un entier
}
delay(5);                                  //préparation pour nouvelles données
if (vitesse >= 0 && vitesse <= 255)        //si vitesse est comprise entre 0 et 255
{
analogWrite(Moteur, vitesse);              //commander le moteur
}
}
```

Version3 : Variation de la vitesse au moyen d'un potentiomètre

On branche maintenant un potiomètre (résistance variable) sur l'une des 6 entrées analogiques(A0 à A5) disponibles sur la carte arduino UNO. Cela nous permet de lire une valeur analogique sous forme d'un entier entre 0 et 1023. En suite en utilisant la fonction spéciale **map()** on peut convertir cet entier en un autre entier compris entre 0 et 255.




```
//Commande d'un moteur à courant continu dans un seul sens de rotation
//et à vitesse variable au moyen d'un potentiomètre

constbyte Moteur=9;
int pot;        //variable dans laquelle on lit la valeur de l'entrée analogique A0
int vitesse;

voidsetup() {
pinMode(Moteur,OUTPUT);
}

void loop () {
pot = analogRead(A0);              //lecture de la valeur donnée par le potentiomètre
vitesse = map(pot,0,1023,0,255);   //convertir la valeur dans pot de 0-1023 à 0-255
analogWrite(Moteur, vitesse);      //commander le moteur
}
```