La plateforme Arduino

Une plateforme de prototypage OpenSource

Xavier Serpaggi

École Nationale Supérieure des Mines de Saint-Étienne

2015

- Rajouter des exemples pour toutes les fonctions Arduino (digital... analog...)
- Rajouter un exemple de code fonctionnel qui relève une valeur de capteur.
- Parler des bibliothèques externes parfois nécessaires.

Matériel

Micro-contrôleurs

Matériel – micro-contrôleurs (μC)

Un μC est principalement composé :

- d'une unité de calcul
- de mémoire
- de modules d'entrée/sortie

Tout ça dans un petit *chip* avec une consommation électrique faible (quelques milli-Watts) et un coût peu élevé.

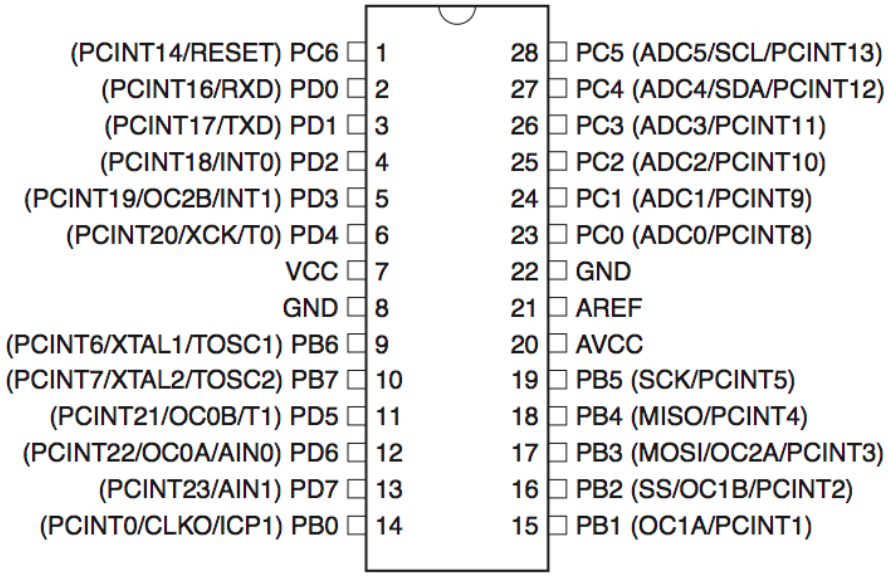
Les principaux fabricants de μC sont Texas Instrument, STMicroelectronics, Atmel.
Nous utiliserons des μC Atmel, de la famille ATMega.

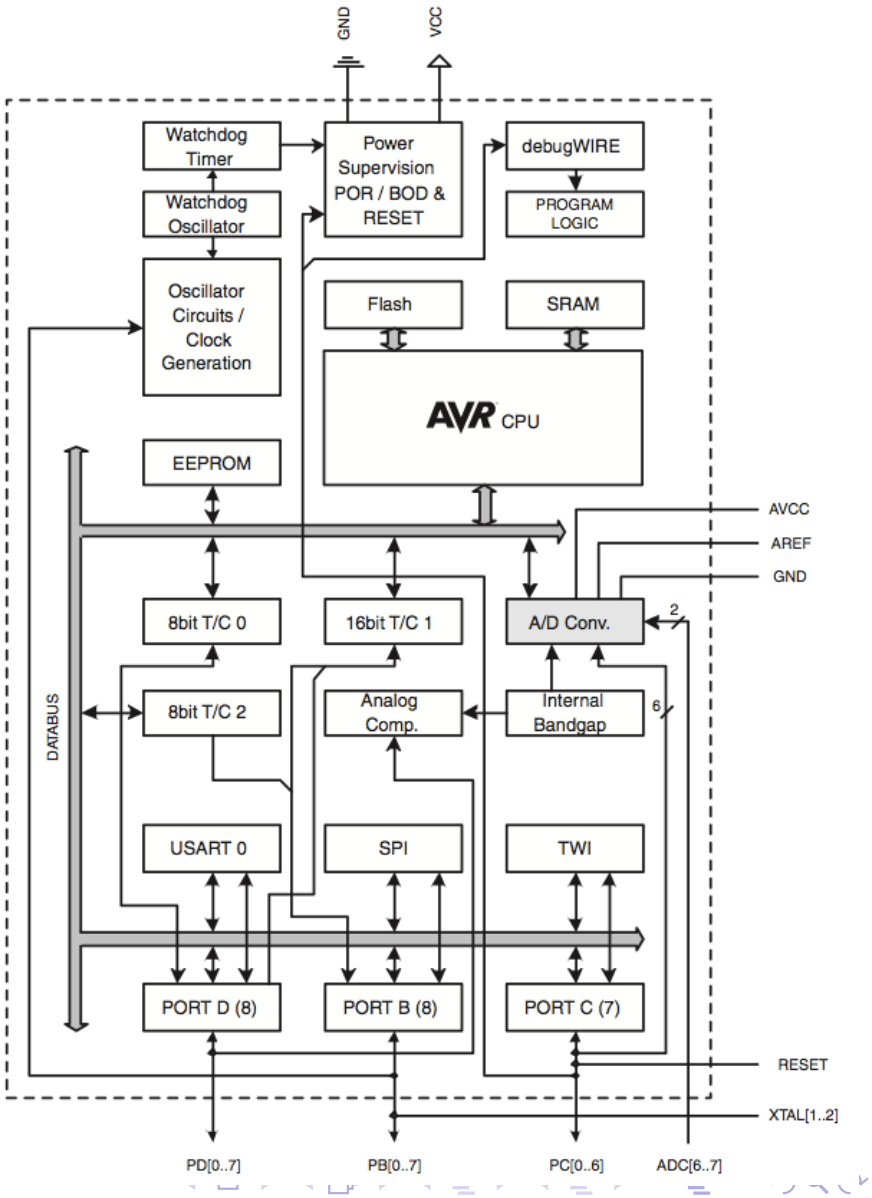
Matériel – les micro-contrôleurs ATMega

Il existe plusieurs μC ATMega, dont l'**ATMega328** et l'**ATMega2560**.
Ce sont tous

- des μC 8 bits *Advanced RISC Architecture* à 16 MHz
- avec de la mémoire (Flash, EEPROM et RAM)
- des lignes d'E/S programmables
- une interface série
- un ADC 10 bits (*Analog to Digital Converter*)
- des *timers*/compteurs 8 et 16 bits
- un comparateur analogique
- ...

Matériel – ATMega328P *block diagram*





Matériel – Architecture mémoire des μC ATMega328

Trois espaces mémoire :

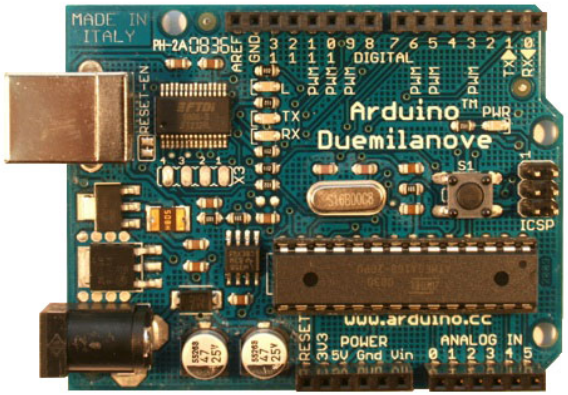
- (32 k) Mémoire Flash (*Flash Program Memory*)
  - *bootloader* dans une section séparée (adresses hautes)
  - organisée en 256 pages de 64 mots de 2 octets (instructions sur 1 ou 2 mots)
  - 10 000 cycles d'écriture/effacement mini
  - accès via le bus SPI
- (registres + 2 k) Mémoire SRAM (*SRAM Data Memory*)
  - organisée en mots de 8 bits
  - stockage des registres : $32 + 64 + 160 = 256$
  - le reste est dédié à l'exécution du programme
- (1 k) EEPROM pour stockage de long terme
  - organisée en 256 pages de 4 mots de 1 octet
  - 100 000 cycles d'écriture/effacement mini
  - accès via le bus SPI

Arduino

Matériel – les cartes Arduino

Plateforme de prototypage construite autour d'un micro-contrôleur Atmel AVR.

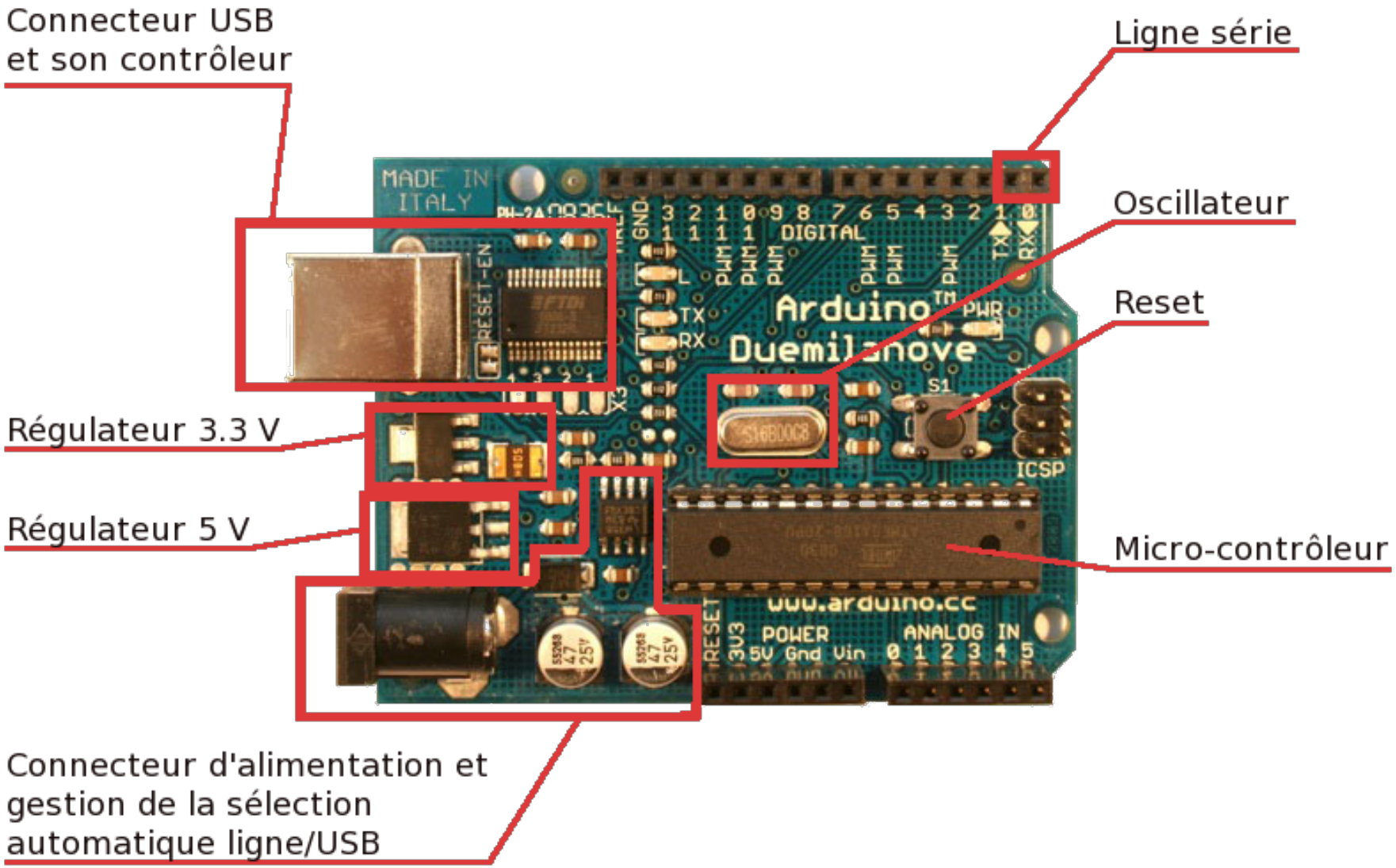
- Destinée au prototypage rapide
- Existe plusieurs variantes
  - Arduino Duemilanove
  - Arduino Uno
  - Arduino mega
  - ...



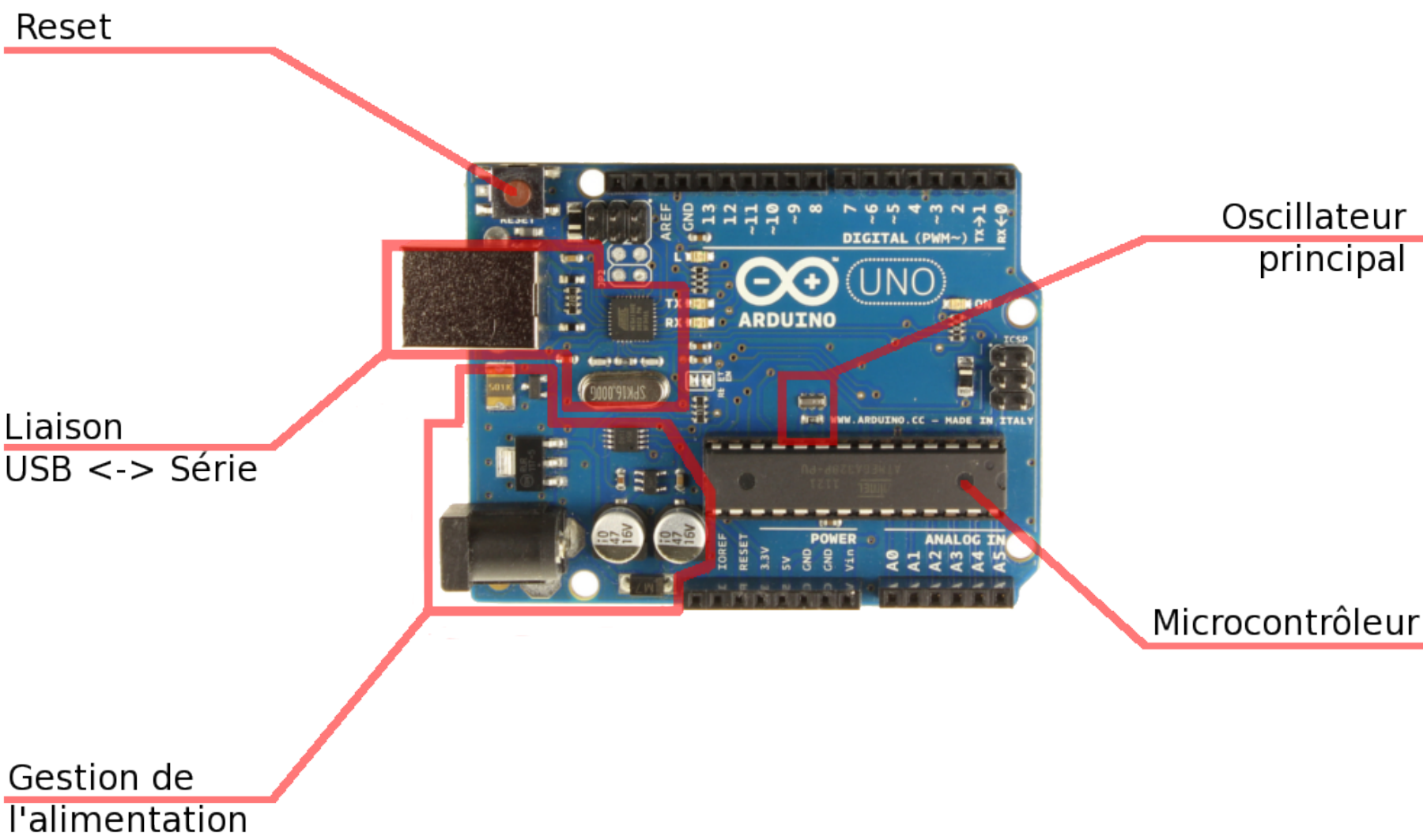
Ces cartes réunissent un μC et tout ce qu'il faut pour le faire fonctionner :

- prise d'alimentation
- connecteurs sur les pattes d'E/S
- *bootloader* (possibilité de graver le programme via USB et lancement de notre code au démarrage)
- connexion avec l'ordinateur (USB, ...)

Matériel – Arduino Duemilanove



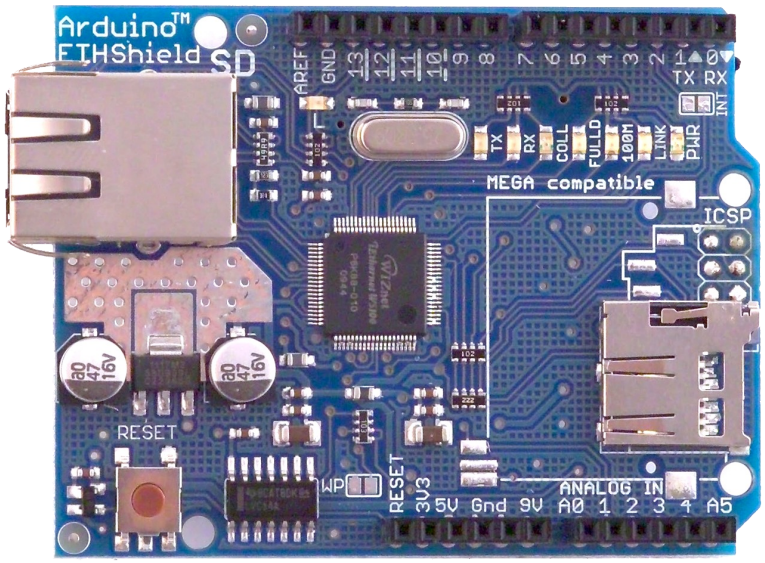
Matériel – Arduino Uno Rev. 3



Matériel – les *shields*

Pour augmenter les fonctionnalités d'une platine Arduino, on peut lui greffer des cartes filles (*shield*) :

- Ethernet
- Wi-Fi
- XBee
- Moteurs
- LCD
- MIDI
- GPS
- ...



Matériel – Arduino

Les *plus* Arduino :

- Architectures matérielle et logicielle **libres**
- Platines prêtes à l'emploi
- API de programmation du μC en C(++) $\rightarrow$ Wiring
- IDE simple d'utilisation $\rightarrow$ Processing

Ces modules sont devenus rapidement populaires dans la communauté des *bricoleurs* (*DIY*) et de nombreuses idées naissent tous les jours.
La popularité des modules Arduino ont fait émerger de nombreux clones et matériels semblables (chipKit, RaspberryPi, Beaglebone, . . .)

Programmation C, environnement de développement

Programmation C – rappels

Programmation C – structure d'un programme

```
#include <stdio.h>

void main()
{
    int i,j ;

    j = 0 ;
    for (i=0 ; i<10 ; i++)
    {
        j = 2*i+j ;
    }

    printf("%d\n", j) ;
}
```

- Chaque **instruction** est terminée par le caractère ;
- Une **fonction** a un nom, des paramètres typés et un type de retour
- La fonction `main()` est obligatoire
- Le comportement d'une fonction est défini dans un **bloc** identifié par des accolades
- Les instructions peuvent être regroupées en bloc si elles doivent être exécutées ensembles
- Les **variables** sont **déclarées** et **initialisées** avant d'être **utilisées**

Programmation C – types de données

- Chaque variable est obligatoirement typée
- Il existe un nombre limité de types
  - `int, long, unsigned int, unsigned long`
  - `float, double`
  - `char`
  - `void`
- Possibilité de créer des tableaux : `int t[10];` définit un tableau contenant 10 entiers, indicés de 0 à 9

Programmation C – préprocesseur

Les directives du préprocesseur :

- permettent de définir des *alias* pour du texte
- sont traitées avant la compilation comme du « *rechercher-remplacer* »
- permettent de créer des macros avec des arguments

Elles sont reconnaissables par le caractère # qui les précède.

```
#define LED_ROUGE 1
#define POUS1 8
#define CAPTEUR_TEMP A2
#define POUS1_ON (digitalRead(POUS1)==HIGH)

#define max(a,b) ((a)>(b)?(a):(b))
```

Programmation C – tests

- Les tests sont à la base des prises de décision simples
- Il y a principalement un type de test :
  - `if ( `*condition*` ) { ... }`
    `else if ( `*condition*` ) { ... }`
    `else { ... }`
- *condition* est fausse si elle s'évalue à 0, elle est vraie sinon
- Les blocs `else if` et `else` sont optionnels
- Les instructions à exécuter pour chaque cas sont regroupées en blocs (`{ ... }`) s'il y en a plus d'une.
- Il existe également la construction `switch` / `case`

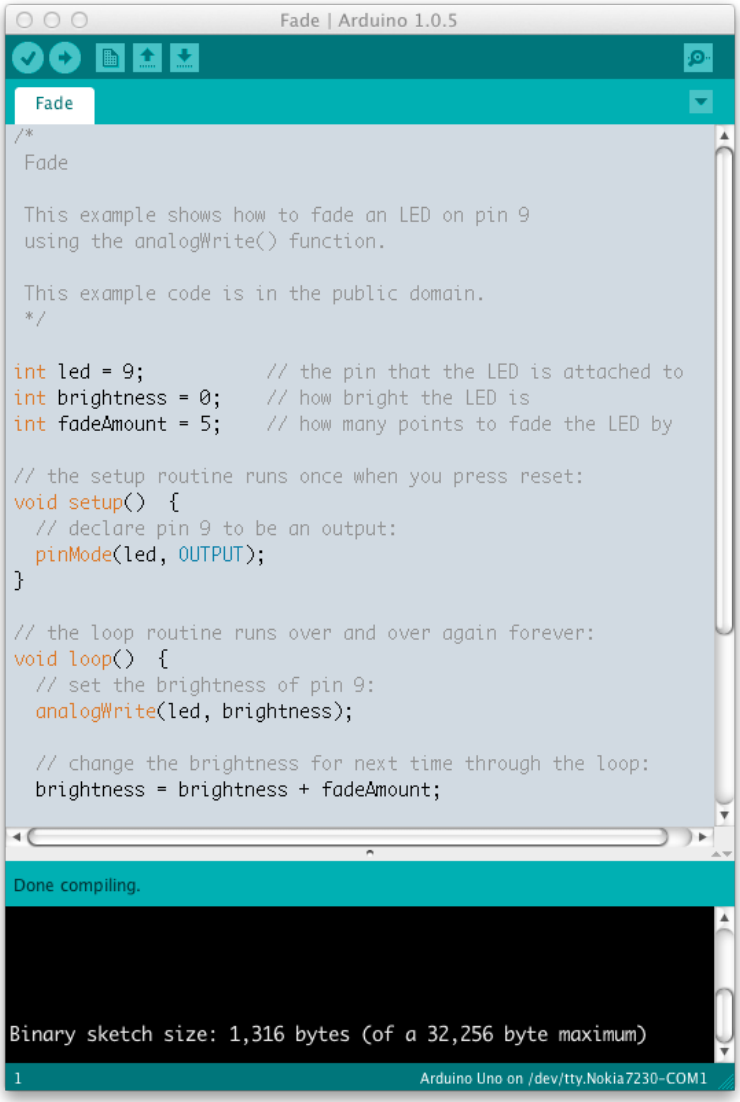
Programmation C – boucles

- Les boucles permettent de répéter un ensemble d'instructions (`{ ... }`)
- 3 types de boucles :
  - `for (` *initialisation* `;` *condition d'arrêt* `;` *incrémentation* `) { ... }`
  - `while (` *condition d'arrêt* `) { ... }`
  - `do { ... } while (` *condition d'arrêt* `);`
- *condition d'arrêt* est fausse si elle s'évalue à 0, elle est vraie sinon
- Seule la boucle `for` explicite l'*initialisation* et l'*incrémentation*
- Dans toutes les boucles, toutes les parties sont optionnelles, selon l'objectif

Environnement de développement (IDE) Arduino

Environnement de développement

- Ces μC disposent d'une architecture RISC avec des jeux d'environ 130 instructions.
- Il existe une API en C++ et un IDE dédié (en Java $\rightarrow$ multi-plateformes)
- Le code est compilé puis, s'il est valide, flashé directement sur le μC.
- Toutes les références de l'API sont installées en local (voir le menu `Aide`) pour votre version de l'IDE



Environnement de développement

Le code est organisé en **sketch**es. Chaque sketch est représenté par un répertoire et au moins un fichier `.ino`.

-  lance la compilation du sketch et affiche les erreurs s'il y en a
-  lance la compilation du sketch et, s'il n'y a pas d'erreur, flashe le programme dans le μC
-  création d'un nouveau sketch
-  ouverture d'un sketch existant
-  sauvegarde du sketch en cours (il n'y a pas de sauvegarde automatique !)
-  ouvre une fenêtre donnant accès à un moniteur série qui permet, entre autre, de visualiser les informations envoyées par la carte

Principes de programmation

- La fonction `main()` est déjà écrite
- Écriture des fonctions `setup()` et `loop()` obligatoire
- Programmation en C/C++ (classes, héritage, ...)

```
int main(int argc, char *argv[])
{
        setup() ;

        for (;;)
                loop() ;

        return 0 ;
}
```

Types de données spécifiques

- 1 octet : `int8_t`, `uint8_t` (`char`, `byte`)
- 2 octets : `int16_t`, `uint16_t` (`int`)
- 4 octets : `int32_t`, `uint32_t` (`long`)
- Il faut faire des tests pour s'assurer du nombre d'octets pour `float`, `int`, `long`

Techniques de programmation

- Favoriser les variables globales et éviter les passages de paramètres dans les fonctions
- Utilisation de constantes (`const` ou `#define`) pour libérer de la mémoire d'exécution
- Taille de la mémoire limitée : il faut dimensionner correctement ses variables (`char`, `int`, `long`, `unsigned` ou pas)

Initialiser les entrées/sorties

Les cartes Arduino sont faites pour nous donner des possibilités d'interaction avec le monde réel.
Pour que ce soit possible il faut initialiser les ports d'entrée/sortie en utilisant la fonction `pinMode(`*`pinNb`*`, `*`mode`*`)`.
mode peut avoir les valeurs `INPUT`, `OUTPUT` ou `INPUT_PULLUP`

- Se fait en général dans la fonction `setup()`
- 6 entrées analogiques qu'il n'est pas nécessaire d'initialiser (ou entrées/sorties numériques qu'il faudra initialiser)
- 14 Entrées/Sorties numériques qu'il est nécessaire d'initialiser

Lecture et écriture sur les entrées/sorties

Utilisation des fonctions :

- `digitalRead(`*`pinNb`*`)` (retourne `HIGH` ou `LOW`) ;
- `digitalWrite(`*`pinNb`*`, `*`level`*`)` avec *`level`* valant `HIGH` ou `LOW` ;
- `analogRead(`*`pinNb`*`)` (retourne un entier compris entre 0 et 1023) ;
- `analogWrite(`*`pinNb`*`, `*`value`*`)` (PWM).

- Se fait généralement dans la fonction `loop()` ou dans toute fonction perso ;
- Possibilité également de faire dans la fonction `setup()` pour :
  - mettre un dispositif connecté dans un état particulier ;
  - (dés)activer la résistance interne de *pull-up* sur les lignes de sortie.

Détails divers

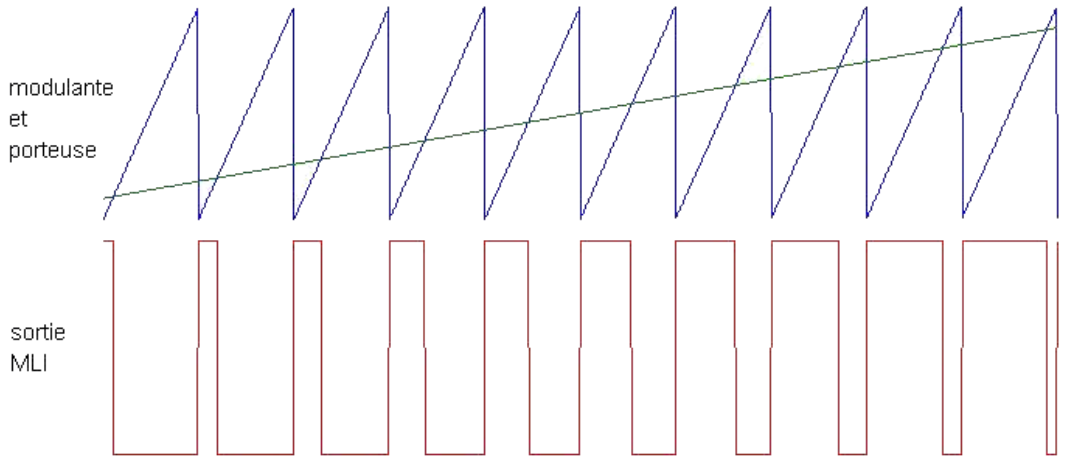
PWM

Pulse Width Modulation

La modulation de largeur d'impulsions (MLI ; en anglais : Pulse Width Modulation, soit PWM), est une technique couramment utilisée pour synthétiser des signaux continus à l'aide de circuits à fonctionnement tout ou rien, ou plus généralement à états discrets.

Le principe général est qu'en appliquant une succession d'états discrets pendant des durées bien choisies, on peut obtenir en moyenne sur une certaine durée n'importe quelle valeur intermédiaire.

(Wikipedia)



Interruptions

Interruptions sur les ATMega328P

Interruption :

> *Signal permettant d'interrompre temporairement le fonctionnement normal d'un programme pour exécuter une routine spécifique.*

26 interruptions disponibles :

- 5 interruptions externes
  - 2 programmables via l'API Arduino, `INT0` et `INT1` (pins 2 et 3)
  - 3 sur changement d'état des pins
- 18 interruptions internes
  - RESET
  - Timers
  - ADC
  - Watchdog
  - Comparateur
  - ...

Interruptions – fonctions utiles

Les interruptions externes peuvent être associées à des fonctions définies par l'utilisateur et permettant de réagir de manière adéquate.

- Activer/désactiver globalement les interruptions : `interrupts()` et `nointerrupts()` ;
- Associer une fonction à une interruption : `attachInterrupt(interrupt, function, mode)` ;
- Supprimer la gestion d'une interruption : `detachInterrupt(interrupt)`.

Le déclenchement des interruptions se fait selon quatre modes principaux :

1. **LOW** : état bas du signal sur la broche
2. **CHANGE** : changement d'état du signal sur la broche
3. **FALLING** : front descendant sur la broche
4. **RISING** : front montant sur la broche

Interface série

Interface série sur un ATMega328P

Il existe une interface série (USART) sur les ATMega328P.

- Utilisée pour les communications inter-modules ou entre le module et l'ordinateur (au travers de l'USB) ;
- Pris en charge par la bibliothèque **Serial** ;
- La ligne série (broches 0 et 1) est exclusive, son utilisation empêche de se servir de ces broches pour autre chose.

Il est donc possible de recevoir des données depuis un module Arduino, mais également de lui en envoyer pour, par exemple, modifier son comportement.

Bus série

Bus série – SPI

SPI : *Serial Peripheral Interface*

- Principe de fonctionnement en maître/esclaves : une ligne `SS` par périphérique ;
- Communications synchrones : une ligne `SCK` commune à tous les périphériques ;
- Deux lignes pour la communications entre maître et esclaves : lignes `MOSI` et `MISO` communes à tous les périphériques.

Il y a une bibliothèque dédiée à ce mode de communication : **SPI Library** qui gère le transfert des données et l'horloge série. Chaque constructeur de périphérique SPI défini ses propres modes de communication.

Bus Série – I^2C

I^2C : *Inter Integrated Circuits*

I^2C, également connu sous le nom de bus TWI (*Two Wire Interface*) nécessite, comme son nom l'indique presque, trois fils pour fonctionner :

1. signal de données : SDA ;
2. signal d'horloge : SCL ;
3. une référence (masse).

Chaque périphérique a une adresse (7 bits) sur le bus et c'est un fonctionnement en maître/esclaves. La bibliothèque permettant de gérer ce bus est **Wire Library**. Elle simplifie grandement la gestion des différents états sur les lignes.

Rappels d'électronique

Potentiel, Intensité

- Différence de potentiel, exprimée en volts (V)
  Se mesure en parallèle de l'élément testé
- Intensité, exprimée en ampères (A)
  Se mesure en série avec l'élément testé

Dans nos montages nous manipuleront principalement du courant continu. Les tensions seront inférieure à 20 volts et les intensités seront de l'ordre de quelques milli-ampères.

Dans le cas de manipulation de courant "secteur" (220V) de grandes précautions devront être prises. En particulier, il faudra séparer les parties haute et basse tension (opto-coupleurs).

L'alimentation des circuits sera notée Vcc et la masse, tension de référence, GND (⏚).

Résistance

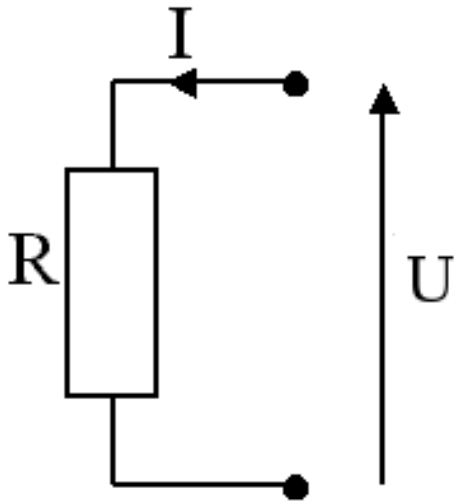
- Composant électronique
- S'oppose à la circulation du courant électrique
- Valeur exprimée en Ohm (symbole Ω) et codifiée par des bandes de couleur ⇒ http://fr.wikipedia.org/wiki/Résistance_(composant)

Il existe des résistances variables dont la valeur change en fonction de données externes (température, luminosité, humidité, action physique, . . .)

Loi d'Ohm

Goerg Ohm (1789 - 1854)

- Loi physique qui lie intensité et tension pour un dipôle électrique donné.
- Courant continu et régime établi : $U = RI$
  U exprimé en Volts, I en Ampères et R en Ohms.

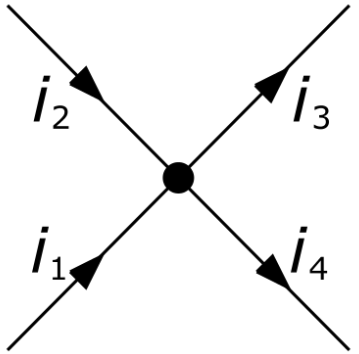


Lois de Kirchhoff

Gustav Kirchhoff (1824 - 1887)

Loi des nœuds

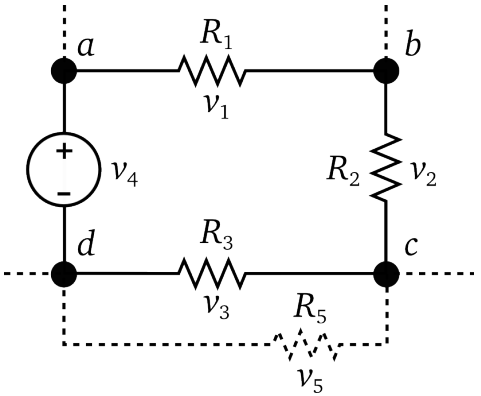
- La somme des intensités des courants qui entrent par un nœud est égale à la somme des intensités qui en sortent.



$$i_1 + i_2 = i_3 + i_4$$

Loi des mailles (simplifiée)

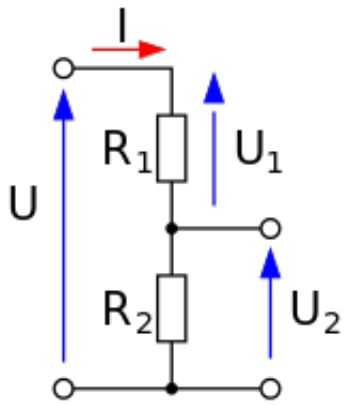
- Dans une maille quelconque d'un réseau, la somme algébrique des différences de potentiel le long de la maille est constamment nulle.



$$V_{ab} + V_{bc} + V_{cd} = V_{ad}$$

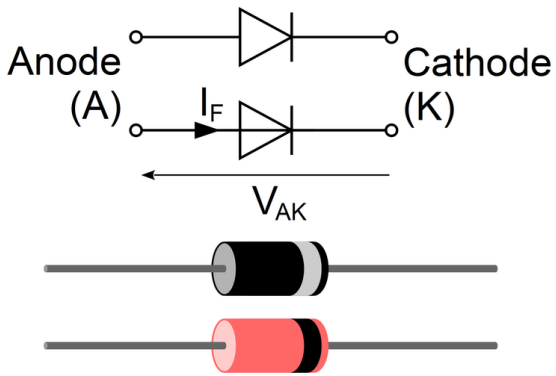
Diviseur de tension

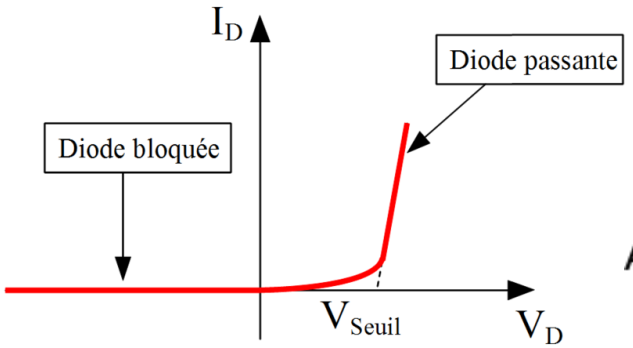
- Montage électronique simple
- $U_2 = U \frac{R_2}{R_1 + R_2}$
- Permet de "décaler" la valeur moyenne d'un signal
- Très utilisé pour lire la valeur de résistances variables (par exemple, en remplaçant R_1 par R_v)



Diode, LED

Diode





- Composant électronique non linéaire
- Ne laisse passer le courant électrique que dans un sens

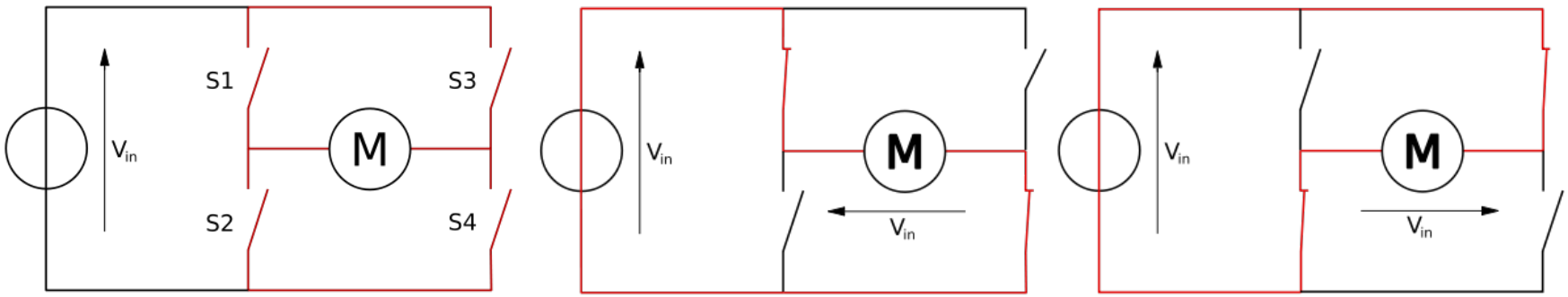
Diode Électro Luminescente (LED)

- Nécessite une tension minimale, un courant pour s'allumer
- Présence d'une résistance de charge pour créer le courant

Moteurs

Matériel – H-Bridge

- Dispositif électronique permettant de piloter une charge
- Permet de faire varier le sens du courant dans la charge



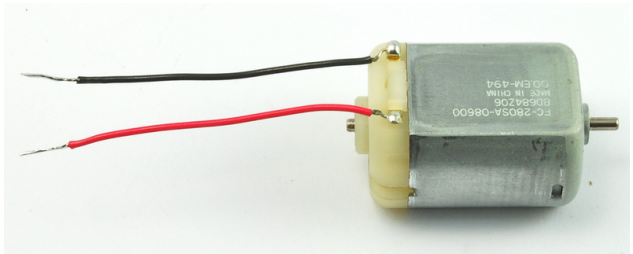
Matériel – Moteurs

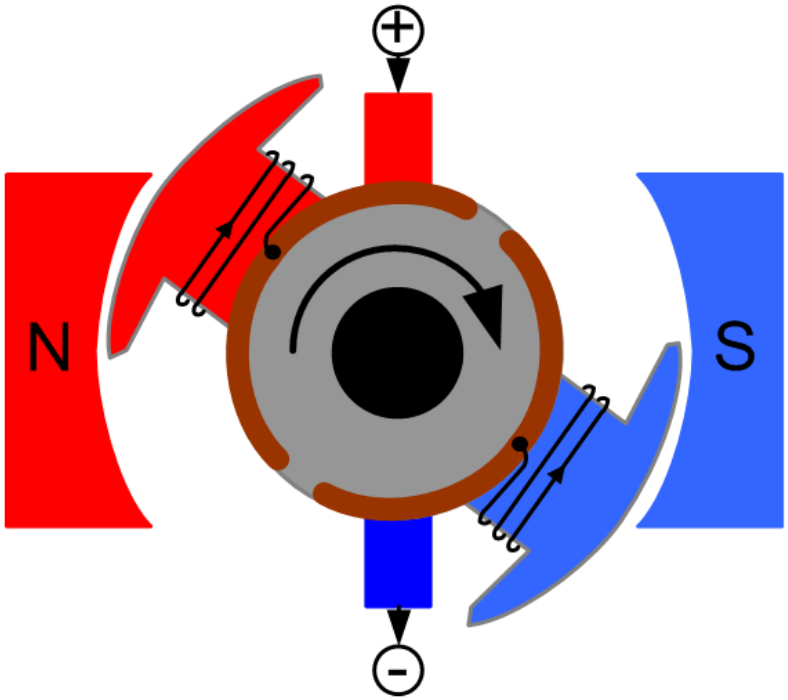
- Dispositif électromécanique
- Rotor et Stator
- Consommation électrique importante
- Courants parasites destructeurs

- Moteur à courant continu (*DC motor*)
- Moteur pas à pas (*stepper motor*)
- Servo-moteur (déclinaison d'un DC, avec de l'électronique)

Matériel – Moteurs à courant continu

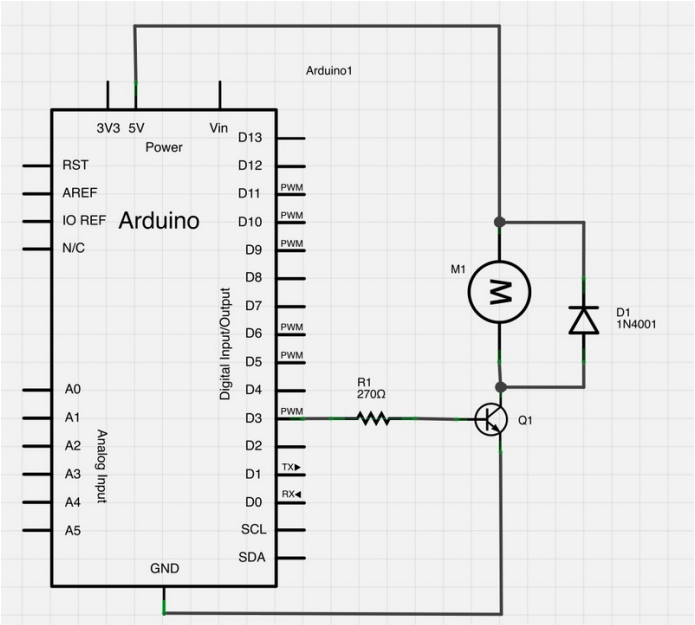
- Le plus simple des moteurs
- Issu de la machine à courant continue inventé par Zénobe Gramme en 1868
- Un stator fixe et un rotor mobile
- Peut-être utilisé comme générateur ou comme moteur
- Moteur :
  - la vitesse dépend de la tension appliquée aux bornes
  - Le sens de la tension va déterminer le sens de rotation
- Générateur : la tension produite dépend de la vitesse imposée





Matériel – Moteurs à courant continu

- Utilisation avec un μC aisée
- Commande de la vitesse par PWM
- Utilisation possible d'un H-Bridge

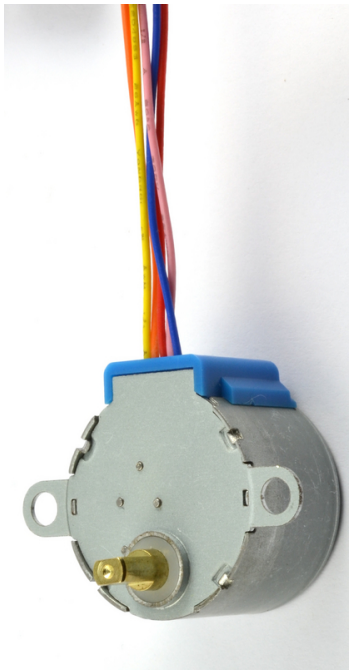


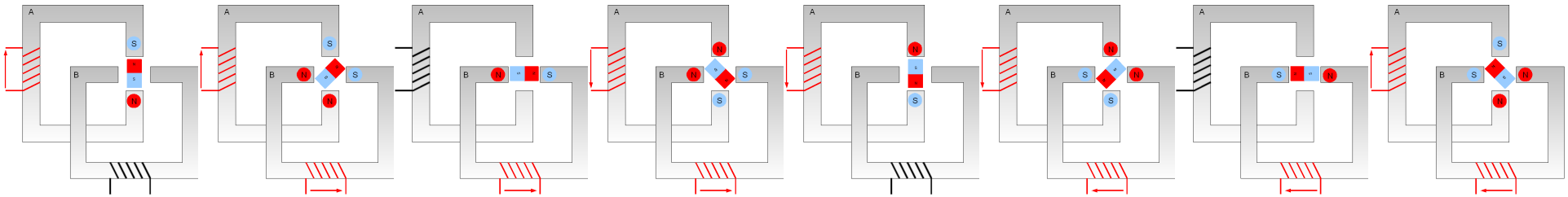
Attention :

- consommation électrique élevée $\rightarrow$ commande par transistor et alimentation autre que le port USB
- courant de retour destructeur $\rightarrow$ nécessité d'avoir un diode de protection

Matériel – Moteurs pas à pas

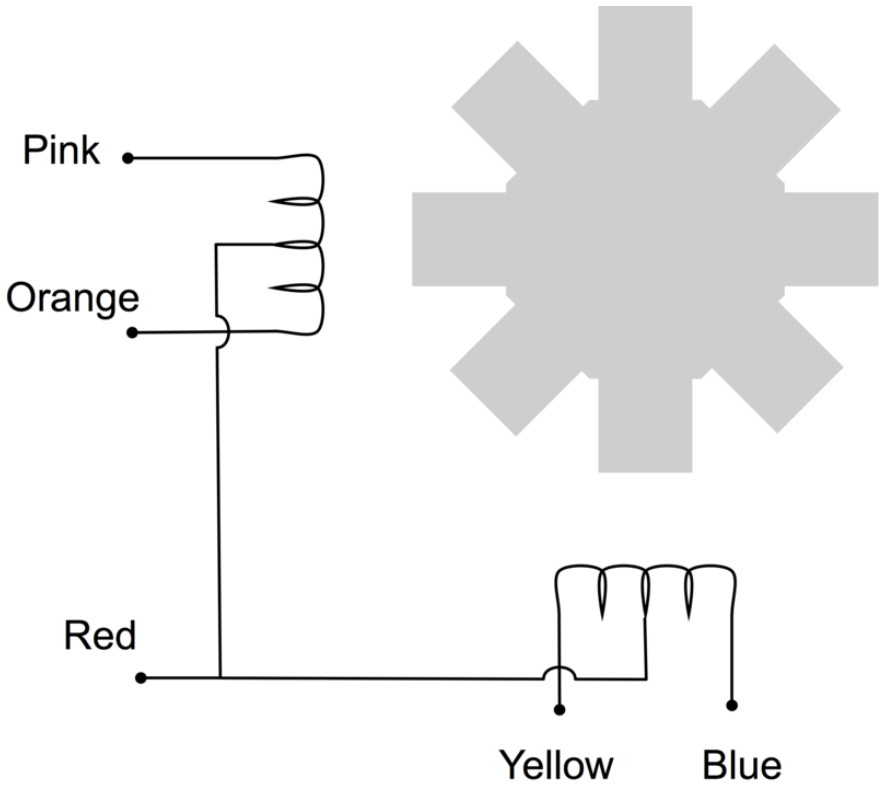
- Peut-être mis dans une position (angle) donnée
- Peut effectuer des rotations complètes
- Ne connaît pas sa position
- Selon la technologie, conserve ou pas sa position après coupure de l'alimentation
- Est déterminé par son nombre de pas (pour faire un tour complet) et par son type de bobinage (uni ou bi polaire)





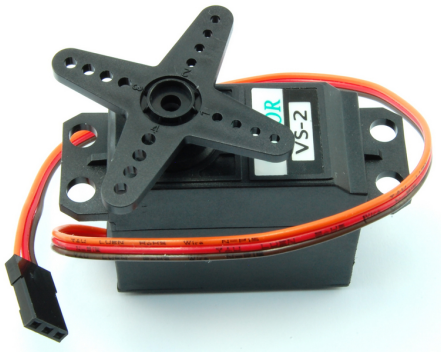
Matériel – Moteurs pas à pas

- Utilisation avec un μC au travers d'un H-Bridge
- Le *jeu* consiste à "allumer" les bonnes bobines au bon moment
- Nous disposons de moteurs bi-polaires (cf. schéma ci-dessous)



Matériel – Servo-moteurs

- Utilisation entre le moteur pas à pas et le moteur DC (moteur pas à pas avec boucle de rétroaction)
- Peut aller à une position donnée à une vitesse donnée
- Ne peut pas toujours faire un tour complet
- Sait maintenir une position si l'on applique un couple de perturbation
- Est adapté à des charges (mécaniques) plus élevées que le moteur pas à pas



Compléments

Autres logiciels

D'autres logiciels permettent d'interagir avec une platine Arduino :

- Processing (`http://processing.org/`)
- Fritzing (`http://fritzing.org/`)
- Eagle (`http://www.cadsoftusa.com/`)

Processing

- IDE basé sur Java et orienté visualisation de données.
- Même interface que l'IDE Arduino (ce dernier est copié sur Processing).
- Même logique de programmation : une fonction `setup()` pour initialiser et une fonction `draw()` pour la boucle d'événements.
- Interface facile avec une platine Arduino via une liaison série pour visualiser sur l'ordinateur ce qui est transmis.
- Exemples de réalisations :
  - Oscilloscope : la platine Arduino reçoit les signaux électriques, les traite et transmet sur la ligne série pour interprétation et affichage par Processing.
  - jeu *pong* : la platine Arduino gère les joueurs (chacun dispose d'un potentiomètre qui va lui permettre de diriger sa raquette) et envoie les valeurs vers Processing qui fait tourner le moteur du jeu.

Fritzing

- Logiciel OpenSource de prototypage de circuits électroniques (Université de Potsdam).
- Orienté vers la plateforme Arduino.
- Permet d'avoir les 3 vues : platine d'essai, schéma électronique, PCB.
- Permet de documenter son projet.
- Bibliothèque de composants limitée, mais extensible.
- De nombreux projets disponibles au téléchargement.

Eagle

- Logiciel professionnel de dessin/validation de circuits électroniques et création de typons.
- Référence dans le domaine.
- Grande bibliothèque de composants qui est en plus, facile à étendre.
- La version gratuite permet de tout faire comme la grande ; elle limite uniquement la taille du PCB.

Bibliographie

- Site web Arduino `http://www.arduino.cc/` avec toutes les fonctions documentées et des exemples
- Page Wikipedia (en) : `http://en.wikipedia.org/wiki/Arduino`
- ***Making Things Talk – Practical Methods for Connecting Physical Objects*, Tom Igoe**
- ***Practical Arduino – Cool Projects for Open Source Hardware*, Jonathan Oxer, Hugh Blemings**
- *Getting Started with Arduino*, Massimo Banzi
- *Arduino Cookbook*, Michael Margolis
- *Arduino notebook*, Brian W. Evans
  `http://muaworkshops.d3cod3.org/programmingBooklet.pdf`

Petit glossaire

- **USART** : *Universal Synchronous and Asynchronous Receiver Transmitter*, module de sérialisation de données destinées à être transmises sur une ligne simple.
- **SPI** : *Serial Peripheral Interface*, bus de communication série
- **I2C** : *Inter Integrated Circuit*, bus de communication série
- **CAN** : *Controller Area Network*, bus souvent utilisé dans le domaine de l'automobile pour dialoguer avec les différents capteurs présents dans un véhicule.
- **EEPROM** : *Electrically-Erasable Programmable Read-Only Memory*
- **ADC** : *Analog to Digital Converter*, module de conversion d'un signal analogique vers des valeurs numériques.
- **RISC** : *Reduced Instruction Set Computer*

Merci