# Informatique embarquée

## Expériences avec une carte Arduino

Philippe.REITZ@LIRMM.FR

décembre 2014

# Plan

1. Présentation générale
   a) Les microcontrôleurs
   b) Leur programmation
2. Les cartes Arduino
   a) Le matériel
   b) L'environnement de programmation
   c) Quelques montages, avec rappels d'électronique
3. Conclusion – perspectives

   Systèmes concurrents

# 1. Présentation générale

- Premiers microprocesseurs
  - 4004 en 1971 (4 bits, <1MHz, Intel)
  - 8080 en 1974 (8 bits, 2MHz, Intel)
  - 6800 en 1975 (8 bits, 2MHz, Motorola)
  - 6502 en 1975 (8 bits, 2MHz, MOS Tech.)
- Le processeur n'intègre sur sa puce que :
  - Les unités logique et de calcul
  - Les registres
  - L' unité de contrôle (décodeur-exécuteur des instructions )
- Sont déportés sur d'autres puces (*chipset*) :
  - Mémoire (bus rapides)
  - Communication avec les périphériques (bus lents)

# 1. Présentation générale

- Idée de SoC (*System On a Chip*)
  - un ordinateur sur une seule et même puce
    - Le processeur
    - La mémoire (données et programme)
    - Le lien au monde extérieur
      - Ports d'entrée/sortie (logiques et analogiques)
      - Chronomètres et gestion d'évènements (interruptions)
- Quelques repères historiques
  - Intel en 1976 : MCS-48 (8 bits)
  - Spécifications ARM, introduites en 1985

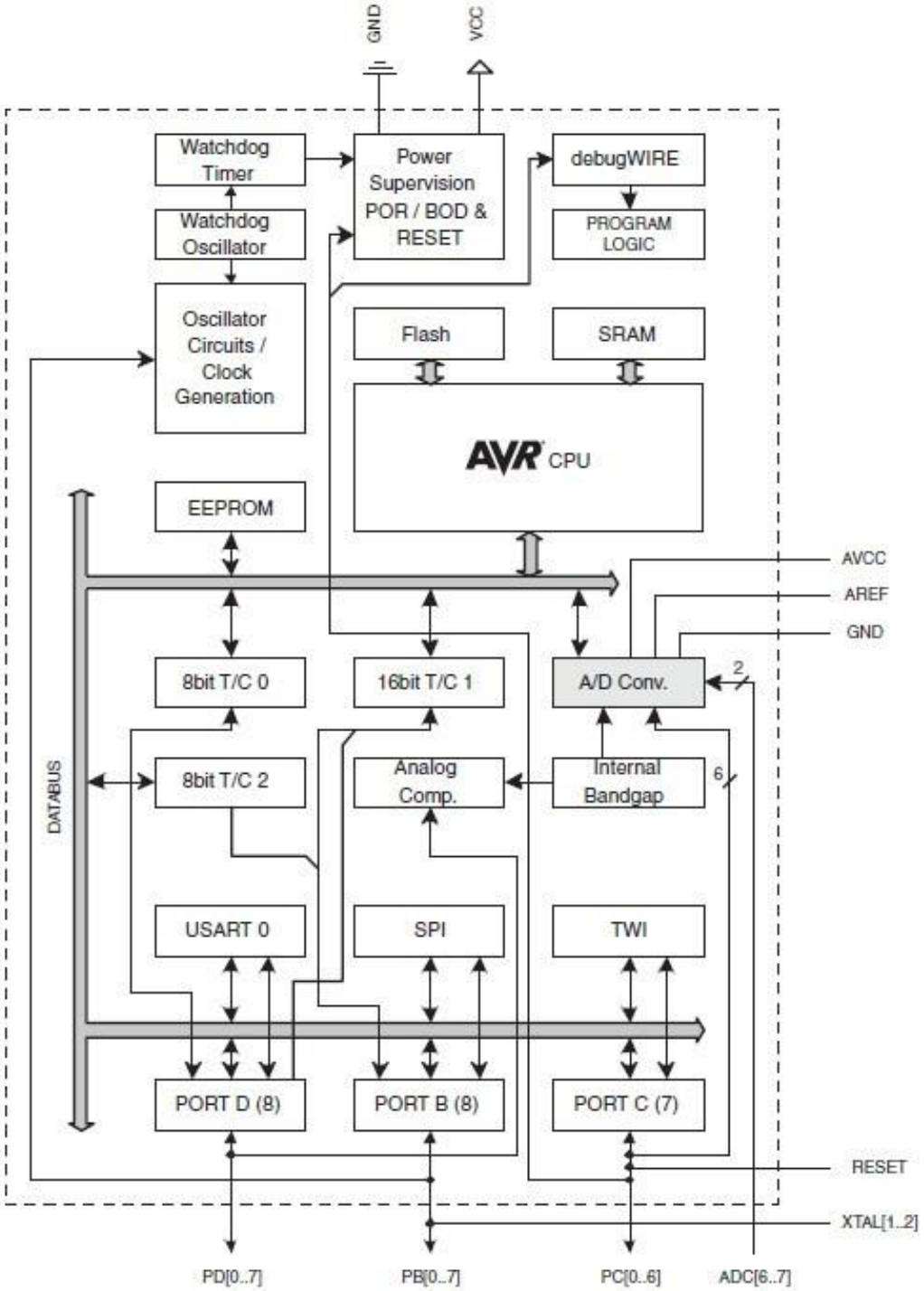
# 1a. Les microcontrôleurs

- Un processeur
  - Architecture (CISC ou RISC)
  - Des registres, d'une capacité donnée (en bits)
- Une mémoire
  - Pour stocker le programme (EEPROM ou Flash)
  - Pour stocker les données (RAM)
- Une horloge
  - Des chronomètres (*timer*), d'une capacité donnée (en bits)
- Des ports d'entrée – sortie
  - Ports numériques (digitaux, logiques) : signaux binaires 0 ou 1
  - Ports analogiques :
    - En entrée : conversion d'une tension en un entier naturel (CAN = conversion analogique / numérique) ; précision exprimée en bits
    - En sortie : conversion d'un entier naturel en une tension (CNA)
    - Attention : toute conversion prend du temps (ex : 100µs sur ATmega328)
- Des interruptions, liées à des évènements particuliers :
  - chronomètre ayant atteint une valeur particulière
  - détection de front d'un signal binaire (front montant = passage de 0 à 1 ; front descendant = passage de 1 à 0)

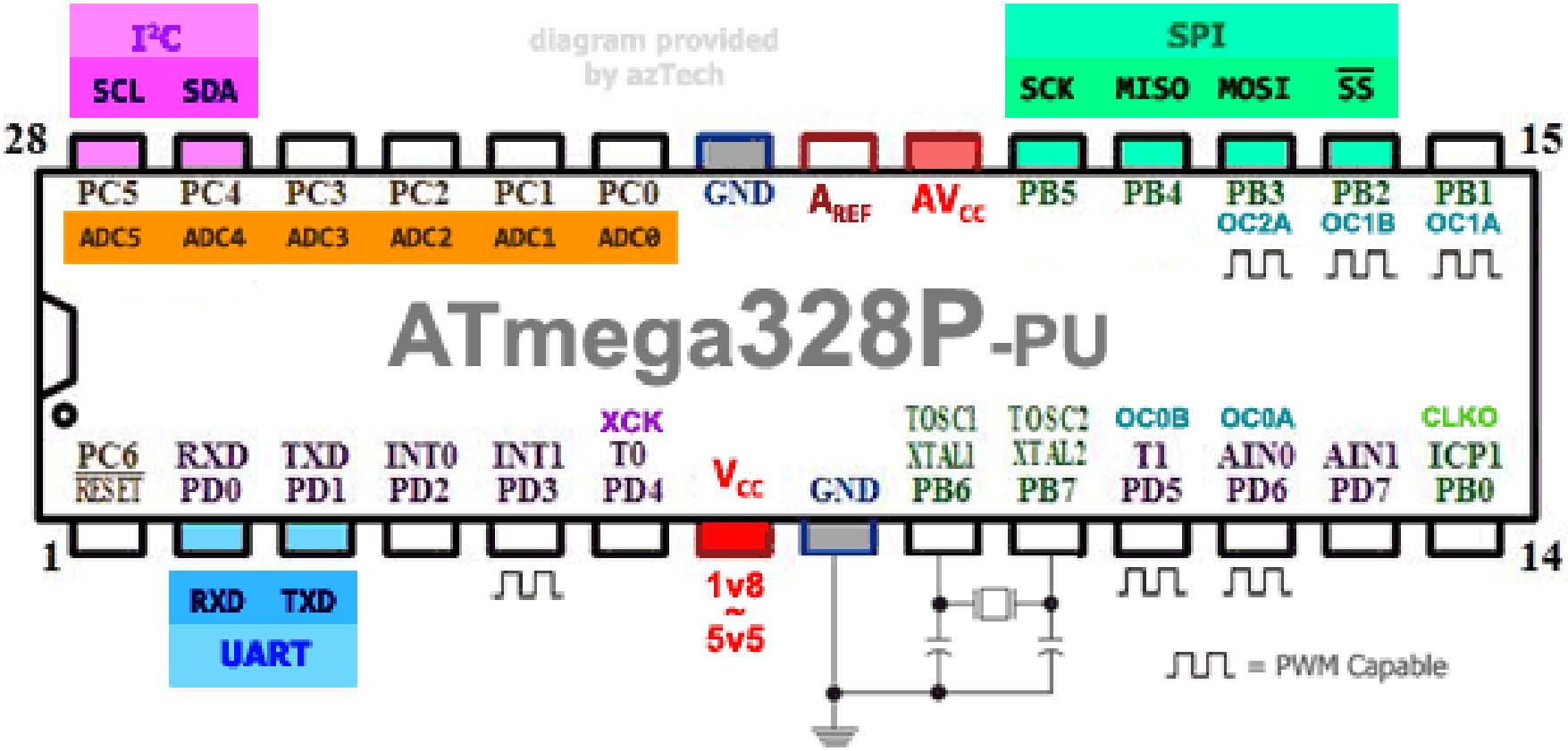
# 1a. Les microcontrôleurs

- Exemple de l'ATmega328P (Atmel, famille AVR)
  - Processeur 8 bits
  - Mémoire :
    - 32k$_i$o de mémoire Flash, 2k$_i$o de SRAM, 1k$_i$o EEPROM
  - Chronomètres :
    - 2 chronomètres 8 bits, 1 chronomètre 16 bits
  - 14 ports numériques programmables en entrée ou sortie
    - dont 6 programmables en sortie MLI/PWM
  - 6 ports analogiques programmés uniquement en entrée
    - 10 bits de précision
  - Liaison série RS-232
  - Coût : 5€

# 1a. Exemple de l'ATmega328P



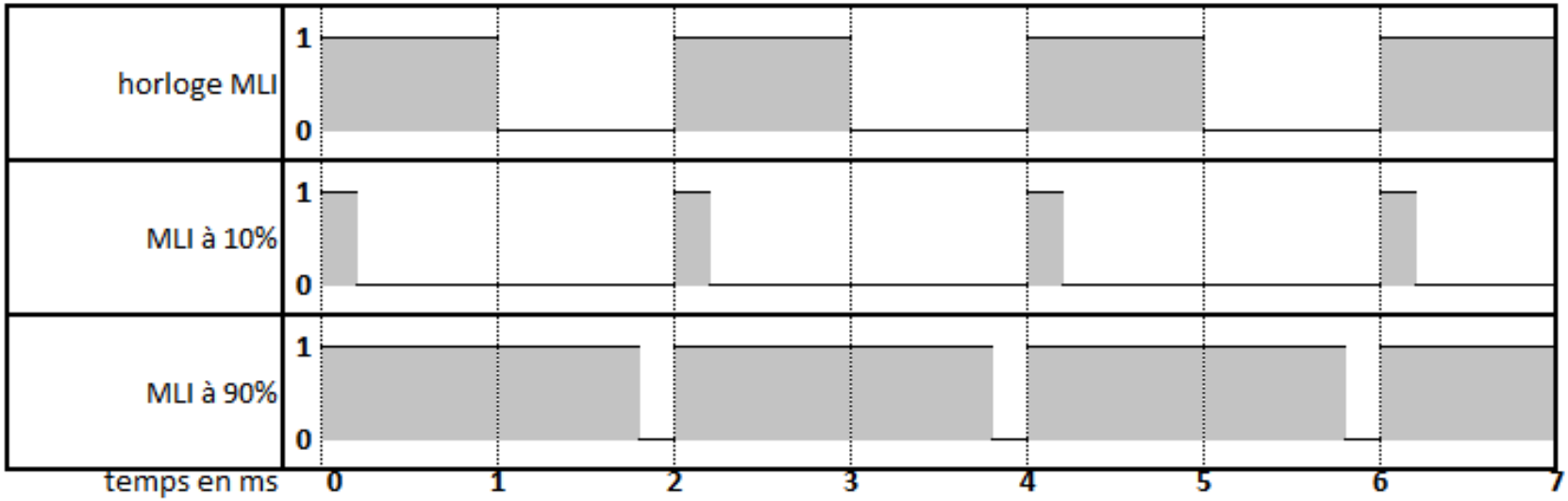
# 1a. Exemple de l'ATmega328P



# MLI/PWM, ou comment simuler de l'analogique avec du numérique pur

- MLI = Modulation en Largeur d'Impulsion

  *PWM = Pulse Width Modulation*




MLI à $\alpha$% $\Rightarrow$ valeur moyenne du signal = $\alpha$

# 1b. Mise en œuvre des microcontrôleurs

- Définition de la partie matérielle
  - Une carte de microcontrôleur est connectée à :
    - des capteurs (périphériques d'entrée) :
      - Interrupteurs, boutons poussoir, potentiomètres, photorésistance, thermistance, autres cartes Arduino ou extensions : Wifi, Ethernet, Bluetooth, I2C, cartes mémoire, détecteurs (ultra-sons, laser, caméra)…
    - des actionneurs (périphériques de sortie) :
      - LED, moteurs, servomoteurs, buzzer, haut-parleur, relais de puissance (transistor, relais), autres cartes Arduino ou extensions : Wifi, Ethernet, Bluetooth, I2C, afficheur LCD, …

    $\Rightarrow$ avoir quelques connaissances en électronique s'avère utile
- Définition de la partie logicielle
  - programmation

# 1b. Programmation des microcontrôleurs

- Langages divers
  - Assembleur lié au microcontrôleur
  - Langages dont les compilateurs peuvent générer du code binaire pour le microcontrôleur
    - C/C++
    - Basic (PIC)
- Processus général :
  - Développement sur un PC, avec émulation ou pas du microcontrôleur et de son environnement
  - Téléchargement du code compilé vers le microcontrôleur

# 1b. Programmation des microcontrôleurs

- Problématique du temps réel :
  - Le délai entre l'acquisition d'une information et son traitement doit respecter des contraintes de temps
  - La puissance de calcul d'un microcontrôleur n'est pas celle d'un PC
- Problématique de la commande de processus :
  - Piloter un ensemble d'actionneurs à partir de données acquises via des capteurs afin de satisfaire une consigne (loi de commande) ne s'improvise pas :
    - Théorie des systèmes (systèmes continus ou discrets, voire hybrides, équations différentielles - transformation de Laplace – ou discrètes – transformation en Z -)
    - Automatique (version moderne de la cybernétique), robotique

# Plan

1. Présentation générale
   a) Les microcontrôleurs
   b) Leur programmation
2. Les cartes Arduino
   a) Le matériel
   b) L'environnement de programmation
   c) Quelques montages, avec rappels d'électronique
3. Conclusion – perspectives

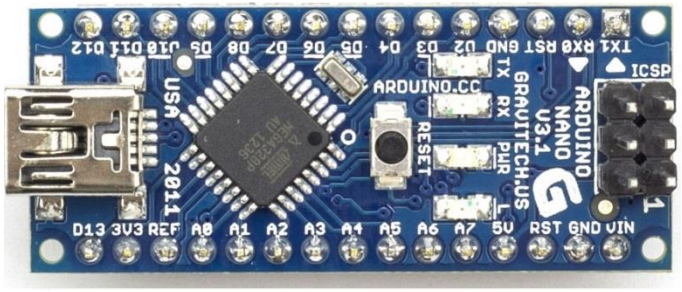
Systèmes concurrents

# 2. Les cartes Arduino

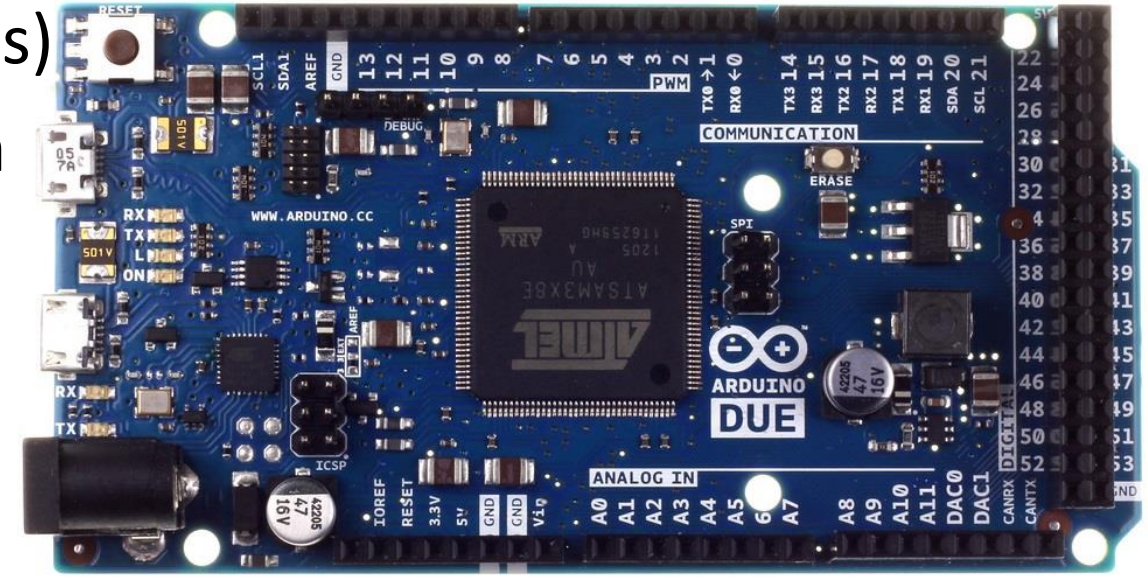
- Origine
  - Projet étudiant en 2005 à l'Institut de Design Interactif d'Ivrée (Italie) ; Massimo Banzi, David Cuartielles, Hernando Barragan
- Cahier des charges
  - Coût le plus faible possible
  - Schémas électroniques des cartes ouverts
  - Environnement de développement libre
    - Interface utilisateur écrite en Java
    - Chaîne de compilation GNU Compiler
  - Site de référence décrivant de nombreux montages
  - Mise en vente de kits et cartes électroniques

# 2a. Les cartes Arduino

- La plus petite : Nano
  - ATmega328
  - 43mm x 18mm, 35€

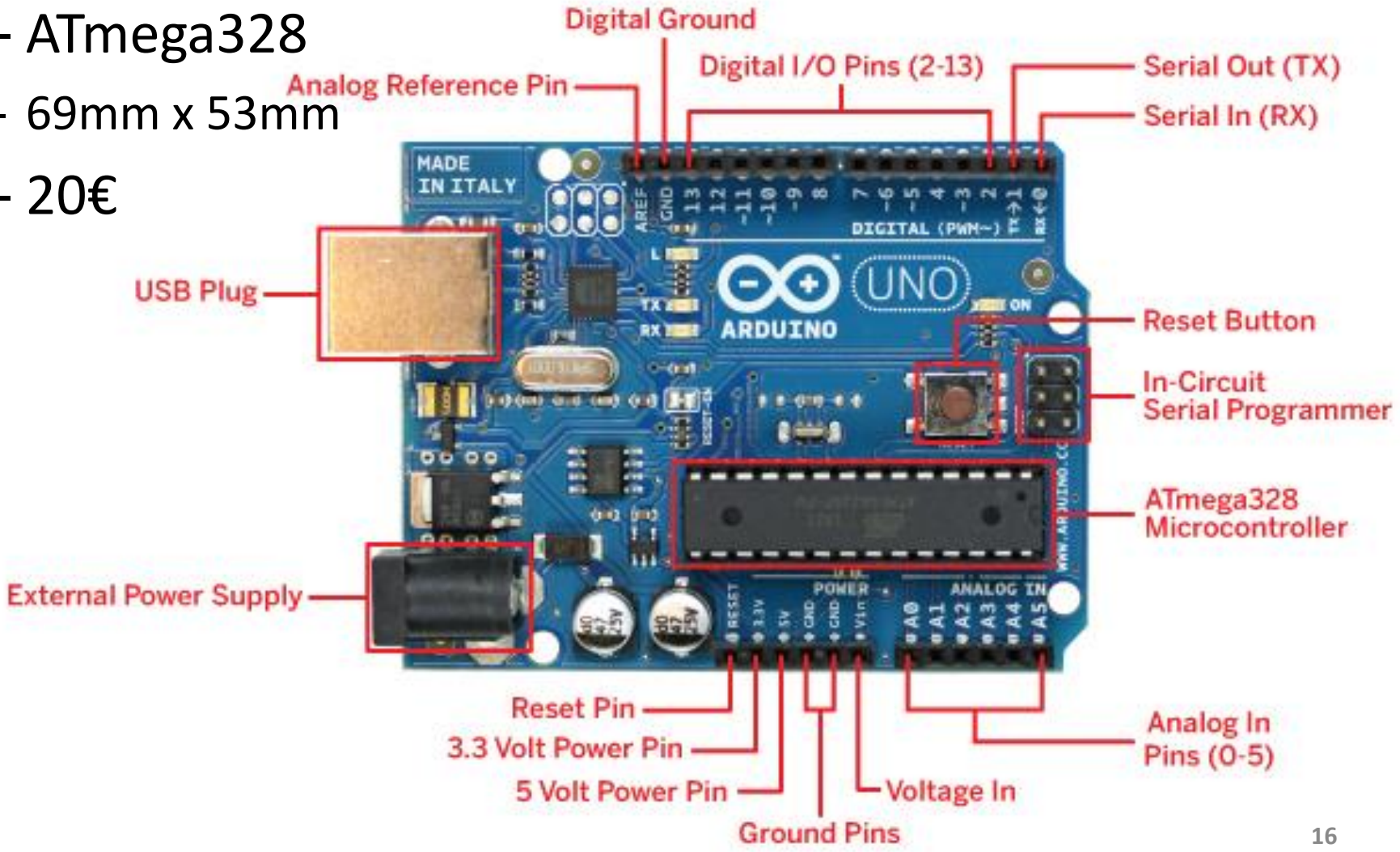



- La plus puissante : Due
  - 32bits, 512k$_i$o Flash, 96k$_i$o SRAM, 54 PN (12 MLI), 12 PA (entrée 12bits)
  - 102mm x 53mm
  - 45€



# 2a. Les cartes Arduino

- Celle exploitée : Arduino Uno
  - ATmega328
  - 69mm x 53mm
  - 20€

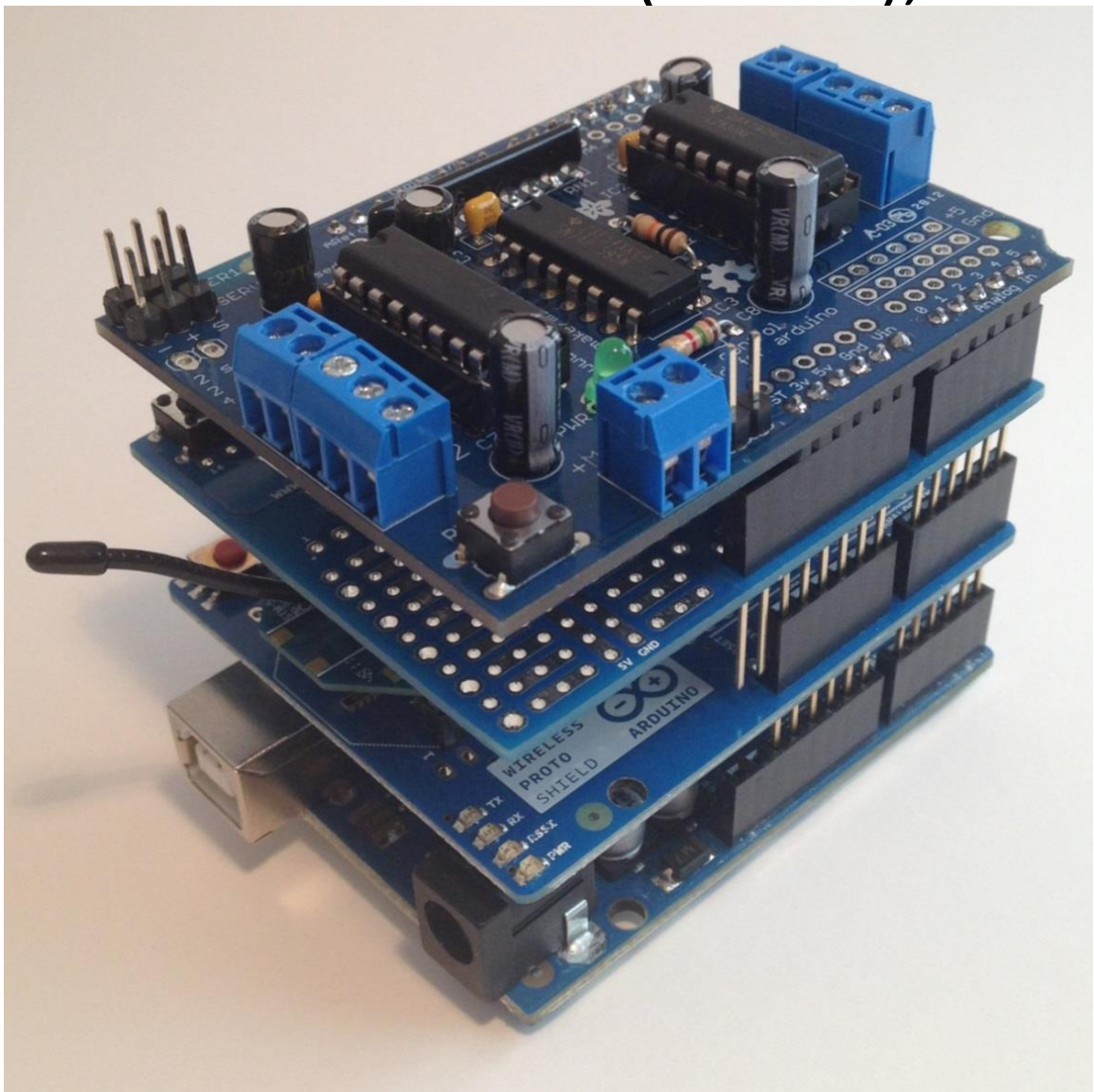


# 2a. La carte Arduino Uno

- Alimentation :
  - 5V via la connexion USB (500mA maximum)
  - Entre 7 et 12V via une alimentation externe (ex : une pile de 9V)
- Puissance :
  - Chaque port en sortie : 40mA maximum
  - Tous ports confondus : 200mA maximum
- Ressources réservées :
  - 2 ports numériques réservés à l'interface série
  - ~½k$_i$o en mémoire Flash réservé au *bootloader*

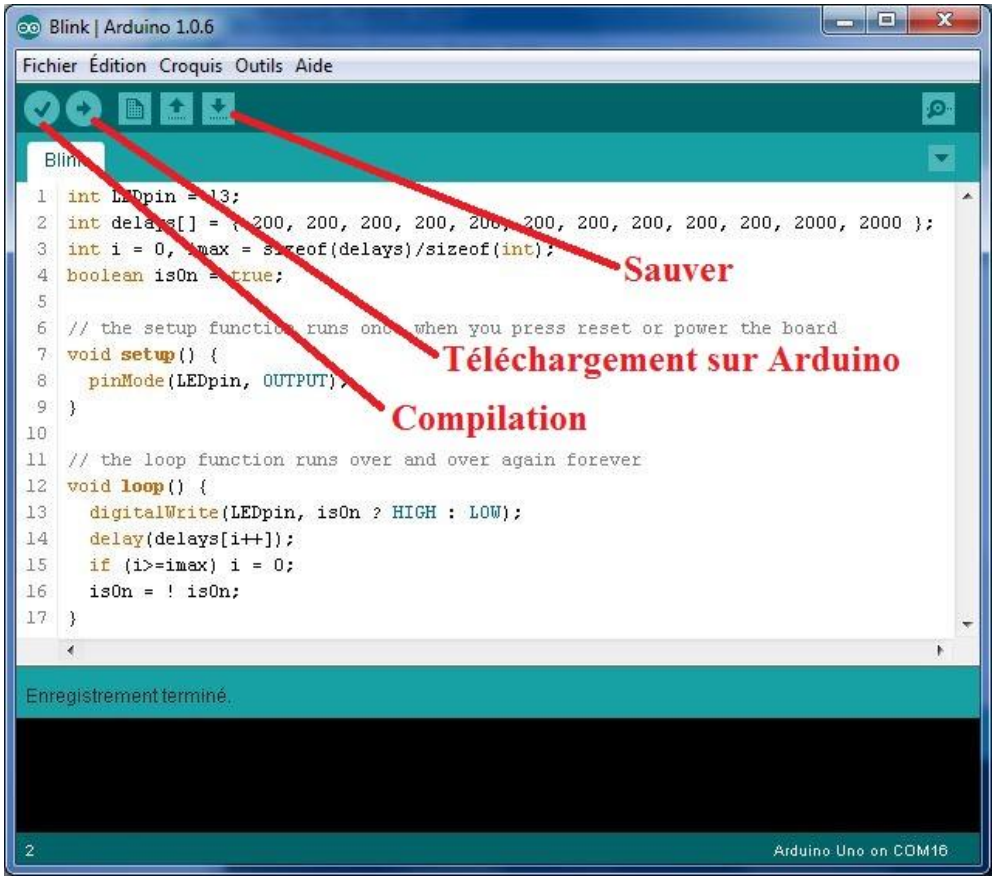
# 2a. Les cartes Arduino

- Nombreuses cartes d'extension (*shields*), empilables :



# 2b. L'environnement de programmation

- Connexion de l'Arduino au PC
  - via une liaison série RS-232 encapsulée dans une connexion USB
- L'environnement de programmation
- Plug'in Eclipse existant (pas testé)



# 2b. Programmer un Arduino

Deux fonctions à définir :

- Fonction `void setup()`
  - Exécutée une seule fois après téléchargement
  - Initialisation du programme
- Fonction `void loop()`
  - Exécutée à l'infini juste après `setup()`
- Un noyau (*bootloader*) pré-chargé gère :
  - La communication série (si exploitée)
  - Le redémarrage du programme si ***reset*** demandé

# 2b. Programmer un Arduino

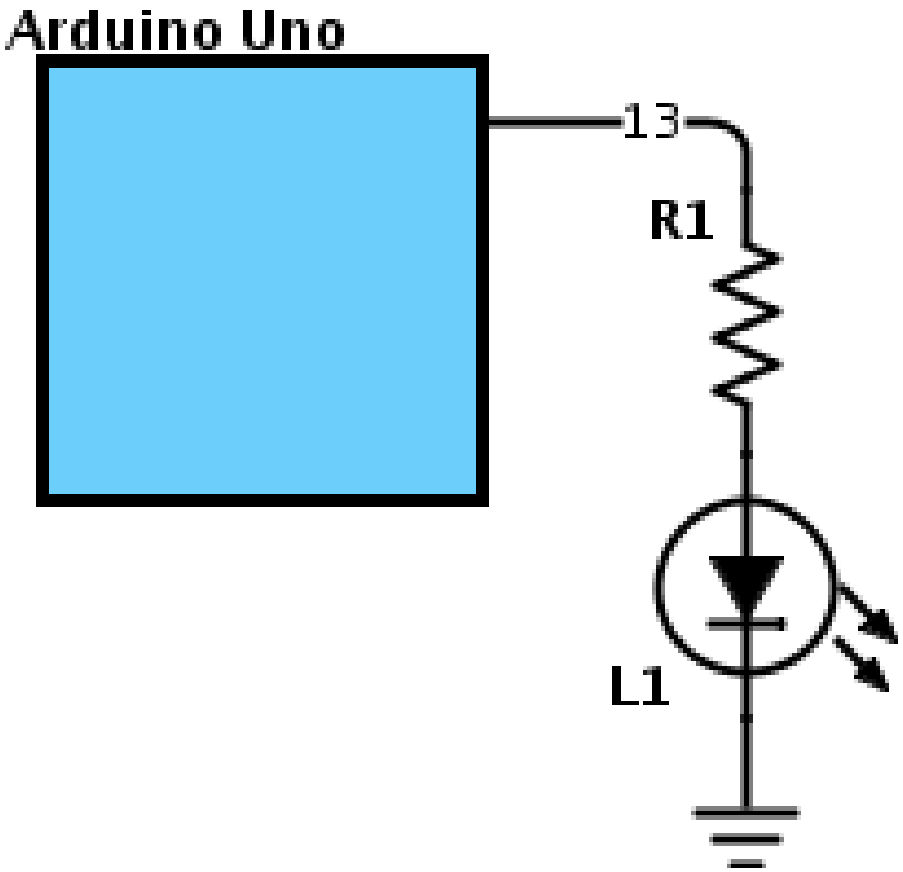
- Une API de base
  - des bibliothèques standard
- Des bibliothèques de contributeurs

# 2c. Analyse de quelques montages

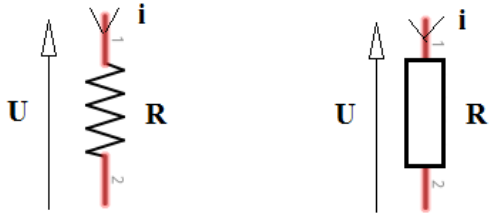
1. Une LED clignotante
2. Deux LED clignotantes, avec des périodes distinctes
3. Une LED pilotée par un interrupteur
   – allumée si bouton poussoir enfoncé
   – Commutation d'état à chaque front montant
4. Une LED dont la luminosité varie
5. Un générateur de son
6. Une étude de cas : voiture de police

# 2c1. Une LED clignotante

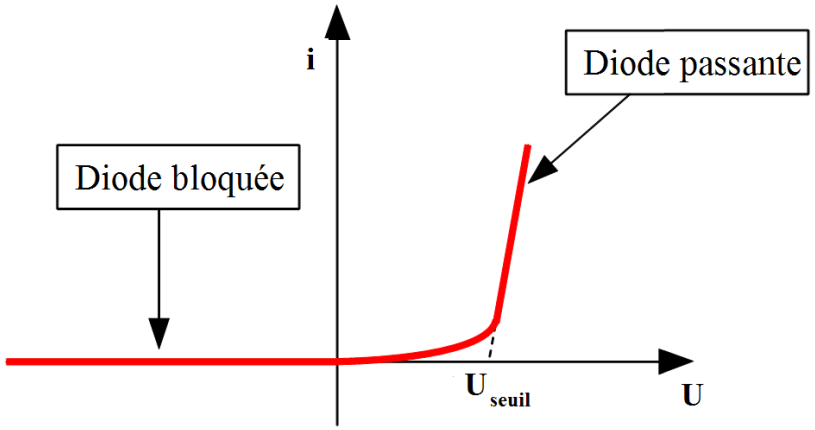
- Montage électronique




**Résistance** $U = R\,i$




**Diode (dont LED)**



# 2c1. Une LED clignotante

- Fonction d'initialisation `setup`

```
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
}
```

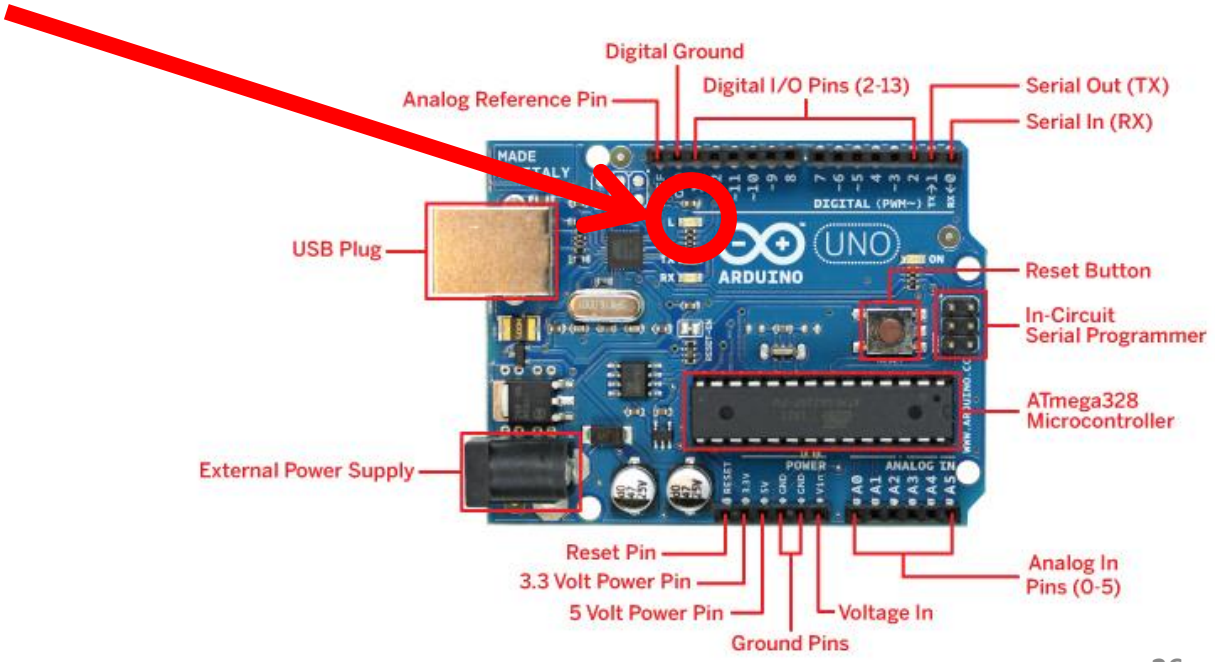
# 2c1. Une LED clignotante

- Fonction de traitement `loop`

```
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
}
```

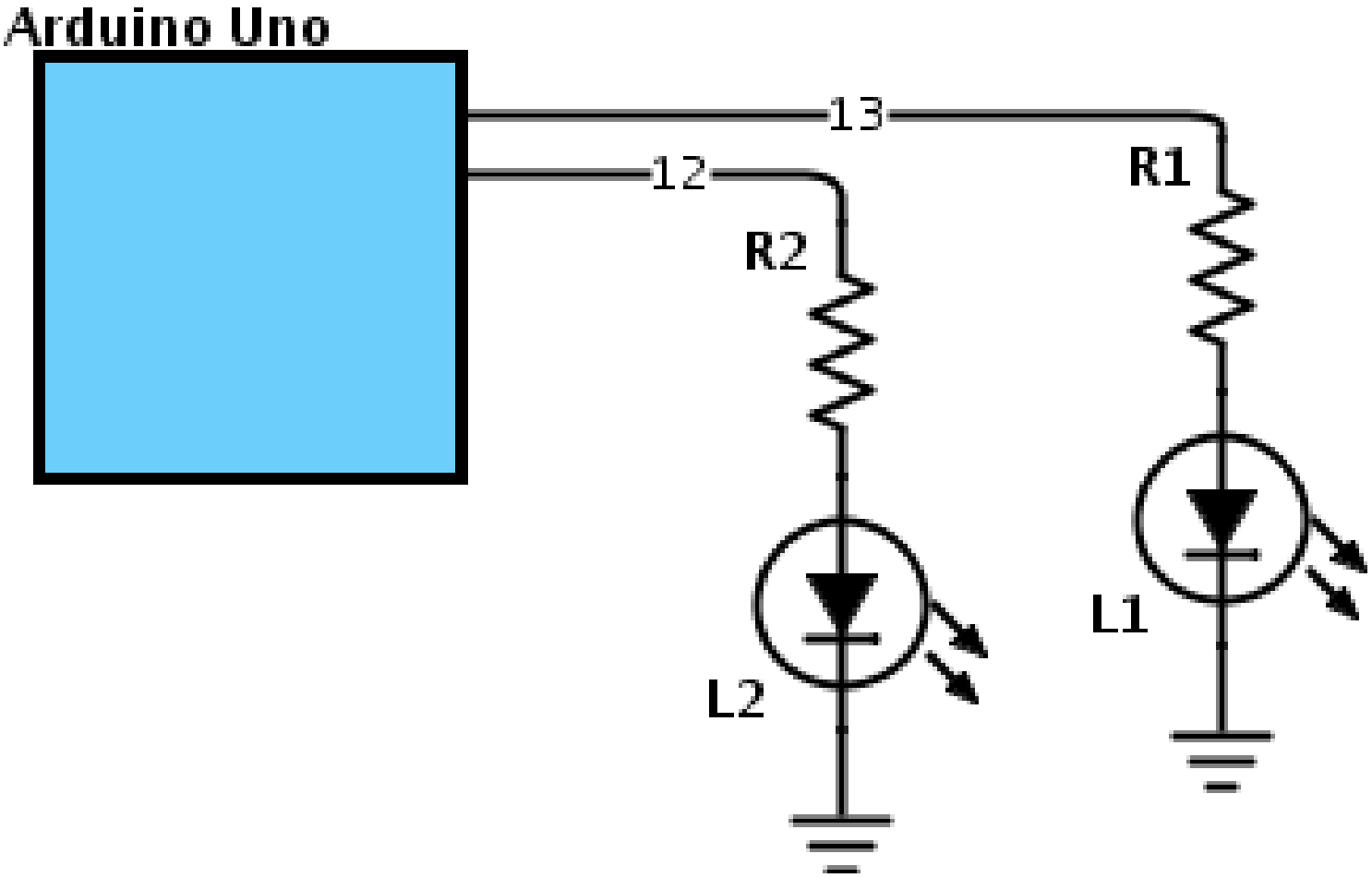
# 2c1. Une LED clignotante

- Montage électronique
  - Inutile de monter une résistance et LED externes
  - Il y a déjà une LED protégée par une résistance pré câblée sur la carte Arduino Uno, reliée à la broche 13



# 2c2. Deux LED clignotantes

- Montage électronique



# 2c2. Deux LED clignotantes

- Fonction d'initialisation `setup`

```
LEDClignotante
   L1 = LEDClignotante(13, 500), // port puis demi-période
   L2 = LEDClignotante(12, 100); //    en millisecondes

void setup() {
   L1.setup();
   L2.setup();
}
```

# 2c2. Deux LED clignotantes

- Fonction de traitement `loop`
  - Une solution avec `delay` ne convient plus

```
void loop()  {
   L1.manage();
   L2.manage();
}
```

# 2c2. Deux LED clignotantes

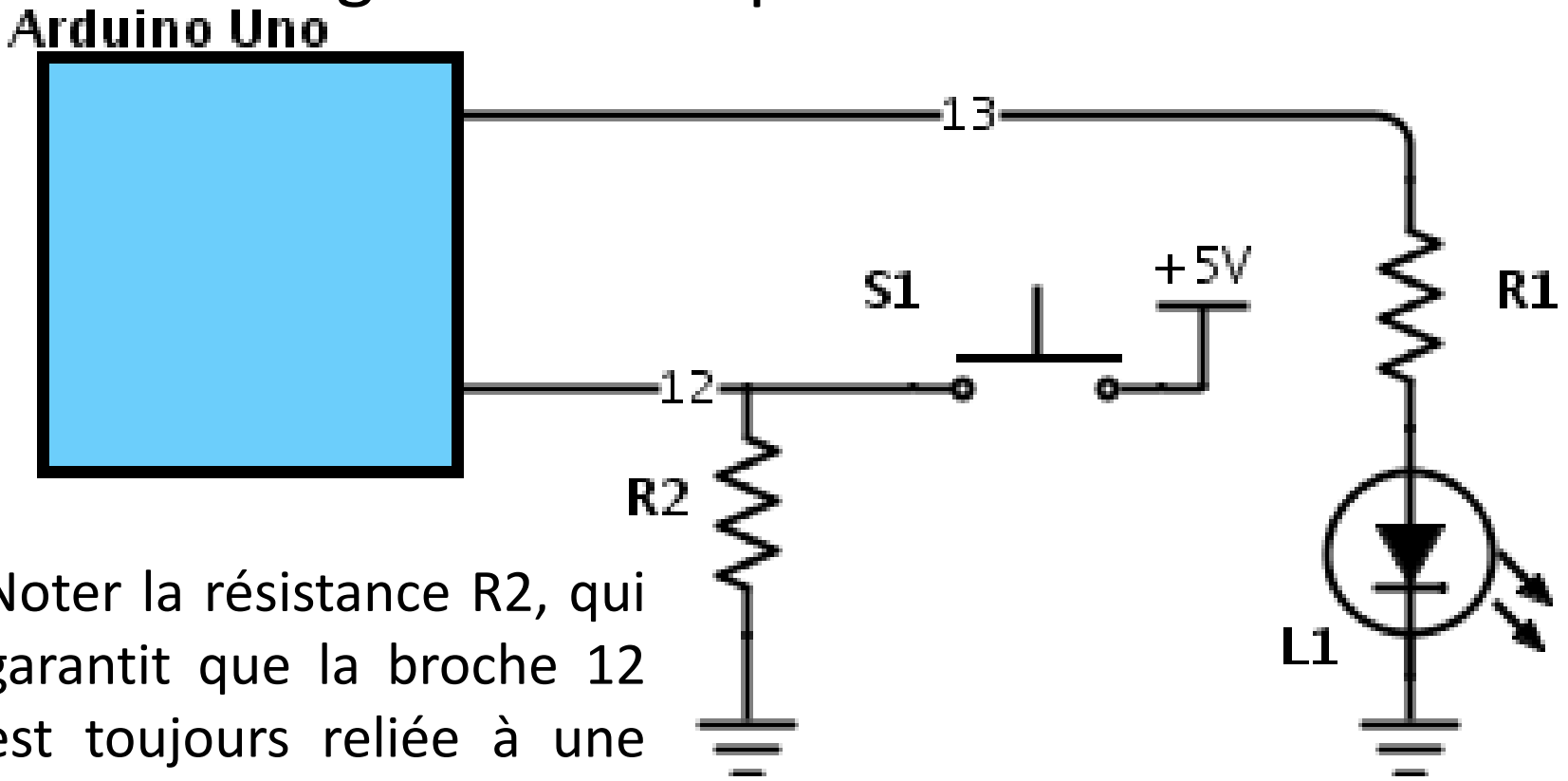
```
class LEDClignotante {
private:
  const int      _pin;    // port numérique associé à la LED
  unsigned int   _delay;  // demi-période de clignotement
  boolean        _isOn;   // état lumineux de la LED
  unsigned long  _clock;  // dernière valeur de l'horloge au dernier chgt d'état
public:
  LEDClignotante(int itsPin, unsigned int itsDelay) :
    _pin(itsPin), _delay(itsDelay), _isOn(false), _clock(0) {}
  void setup() { // méthode d'initialisation
    pinMode(_pin, OUTPUT);
    digitalWrite(_pin, _isOn ? HIGH : LOW);
    _clock = millis();
  }
  void manage() { // méthode de gestion du clignotement
    unsigned long currentClock = millis();
    if (currentClock-_clock>_delay) {
      _clock = currentClock;
      _isOn = ! _isOn;
      digitalWrite(_pin, _isOn ? HIGH : LOW);
    }
  }
};
```

# 2c2. Deux LED clignotantes

- Il existe d'autres solutions :
  - Exploiter les interruptions et les chronomètres
  - S'appuyer sur des sources de temps extérieures
    - Purement analogiques : oscillateurs
    - Purement logiques

# 2c3. Une LED pilotée par un bouton poussoir

- Montage électronique




Noter la résistance R2, qui garantit que la broche 12 est toujours reliée à une tension de référence

# 2c3. LED pilotée par un bouton poussoir

- Fonction d'initialisation `setup`

```
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
}
```

# 2c3. LED pilotée par un bouton poussoir

- variante 1 : LED allumée tant que bouton enfoncé
  - Fonction de gestion `loop`

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

# 2c3. LED pilotée par un bouton poussoir

- variante 2 : la LED change d'état à chaque fois que le bouton est enfoncé puis relâché (front montant)
  - Ajout de deux variables d'état :
    - `étatL1` indique si la LED est éteinte (`LOW`) ou allumée (`HIGH`) ; initialisée à `LOW`
    - `étatS1` indique si le bouton poussoir est enfoncé (`HIGH`) ou relâché (`LOW`) ; initialisée à `LOW`

# 2c3. LED pilotée par un bouton poussoir

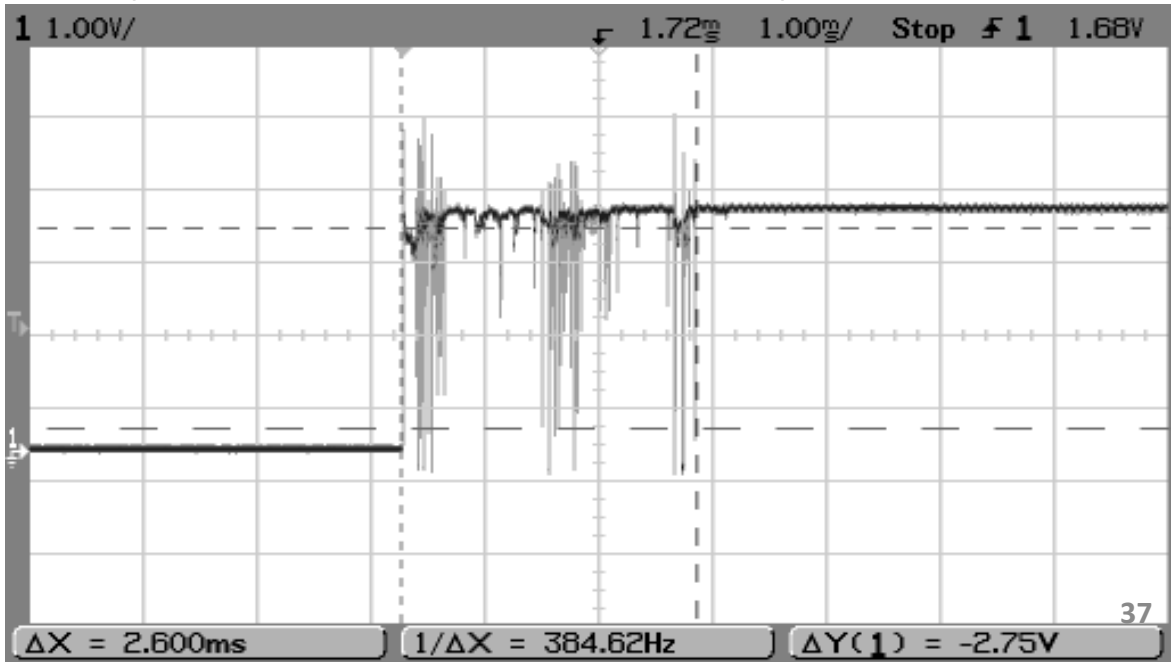
- variante 2 (suite)
  - Fonction de gestion `loop` naïve

```
inline int complement(int level) { // level = HIGH ou LOW
  return level==HIGH ? LOW : HIGH; }

void loop() {
  int etatCourantS1 = digitalRead(brocheS1);
  if (etatS1==LOW && etatCourantS1==HIGH) {
    // front montant détecté sur le bouton poussoir
    etatL1 = complement(etatL1);
    digitalWrite(brocheL1, etatL1);
  }
  etatS1 = etatCourantS1;
}
```

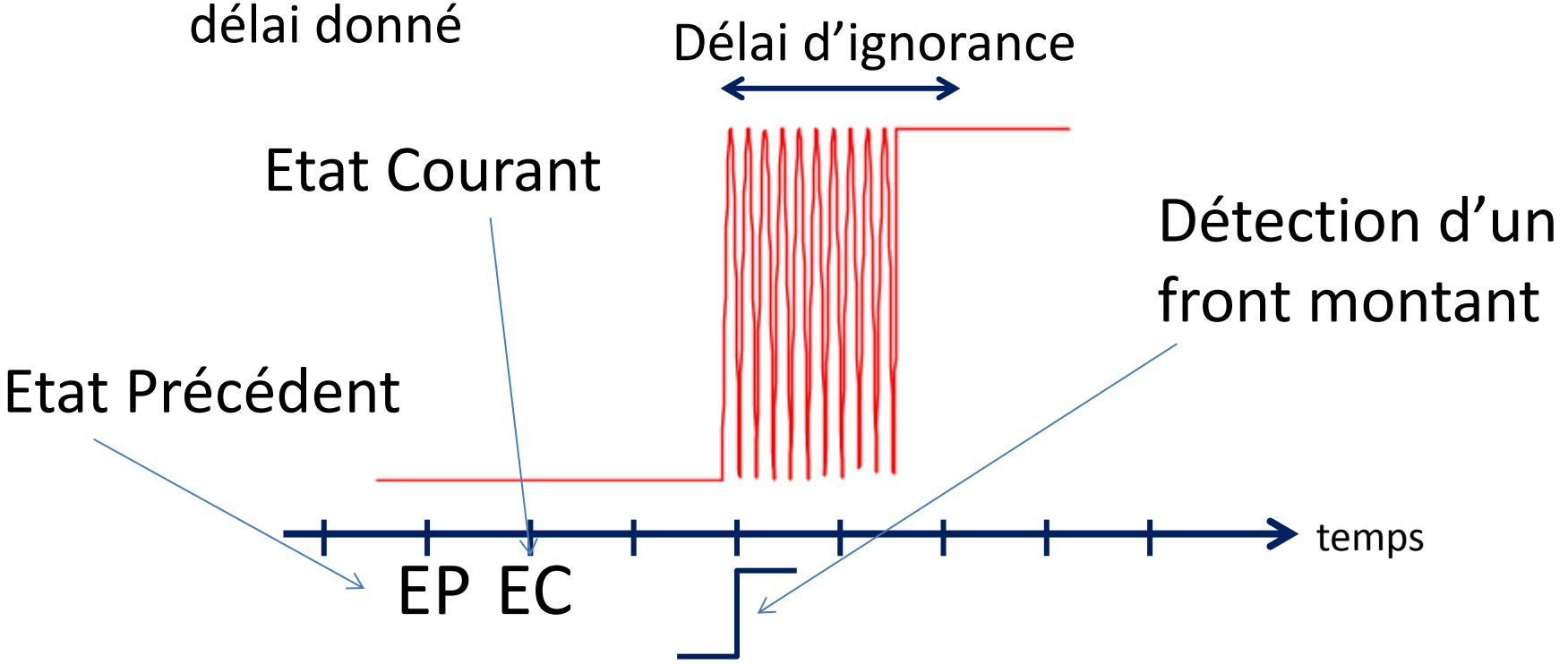
# 2c3. LED pilotée par un bouton poussoir

- variante 2 (suite)
  - Constat : le montage ne fonctionne pas bien
    - À chaque fois que le bouton poussoir est enfoncé puis relâché, la LED s'allume ou s'éteint de façon aléatoire
  - Explication : phénomène du rebond
    - Les contacts électriques d'un interrupteur rebondissent l'un sur l'autre lorsqu'ils entrent en contact (pendant quelques ms).



# 2c3. LED pilotée par un bouton poussoir

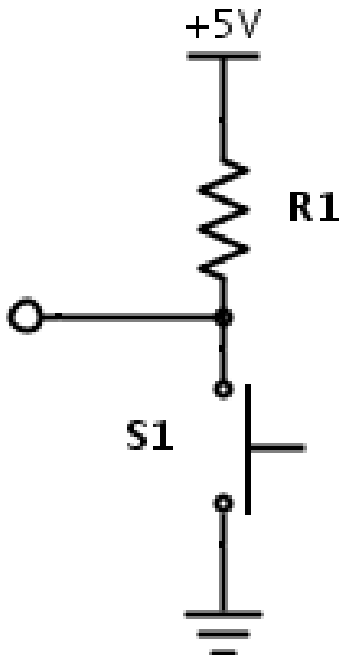
- variante 2 (suite)
  - Solution 1 : rebond géré au niveau logiciel
    - détectant le premier front montant
    - puis qui ignore les changements qui suivent pendant un délai donné



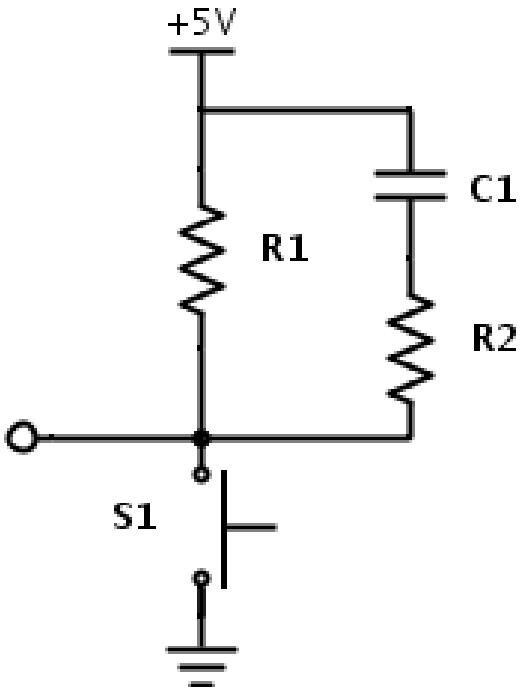
# 2c3. LED pilotée par un bouton poussoir

- variante 2 (suite)
  - Solution 2 : rebond géré au niveau électronique

Sans protection anti-rebond

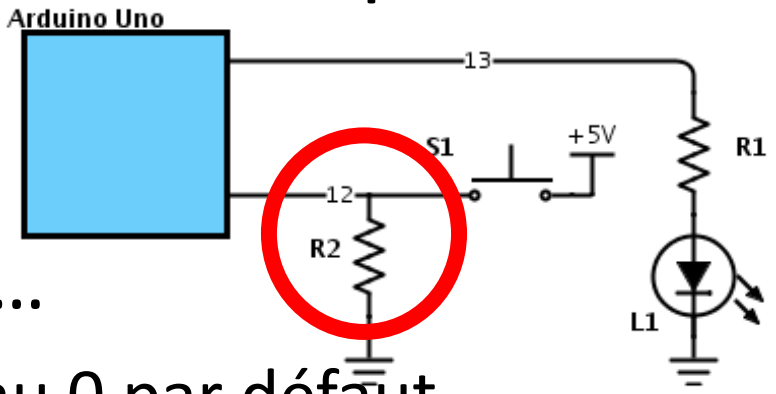



Avec protection anti-rebond




1. Bouton relâché
   C1 déchargé : tension nulle à ses bornes, donc 5V sur la broche
2. Bouton enfoncé
   C1 se charge via R2 : une fois chargé (supposé rapide), tension de 5V à ses bornes ; 0V sur la broche
3. Bouton vu comme relâché (rebond)
   C1 se décharge via R1+R2 (supposé lent) ; tension passant de 0V à 5V sur la broche

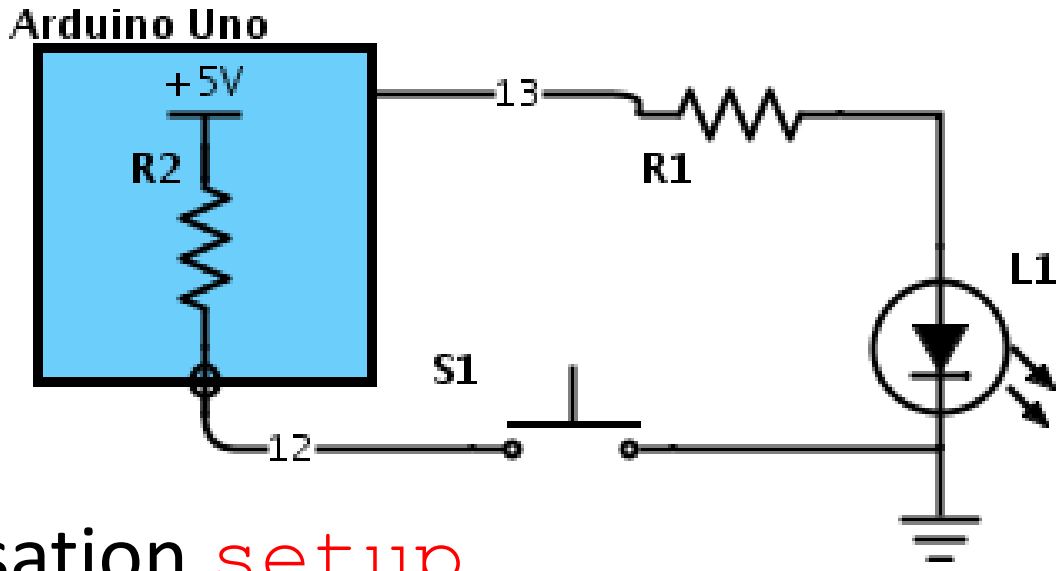
# 2c3. Une LED pilotée par un bouton poussoir




- À propos de la résistance R2 associée au bouton poussoir…
  - Si la résistance assure un niveau 0 par défaut (connectée à la masse), elle est dite :
    - **résistance de rappel** (*Pull-down resistor*)
  - Si elle assure un niveau 1 par défaut (connectée au 5V), elle est dite :
    - **résistance de tirage** (*Pull-up resistor*)
- Le ARmega328 dispose pour chaque port numérique d'une résistance de tirage activable ou pas par programme.

# 2c3. Une LED pilotée par un bouton poussoir

- Le montage devient



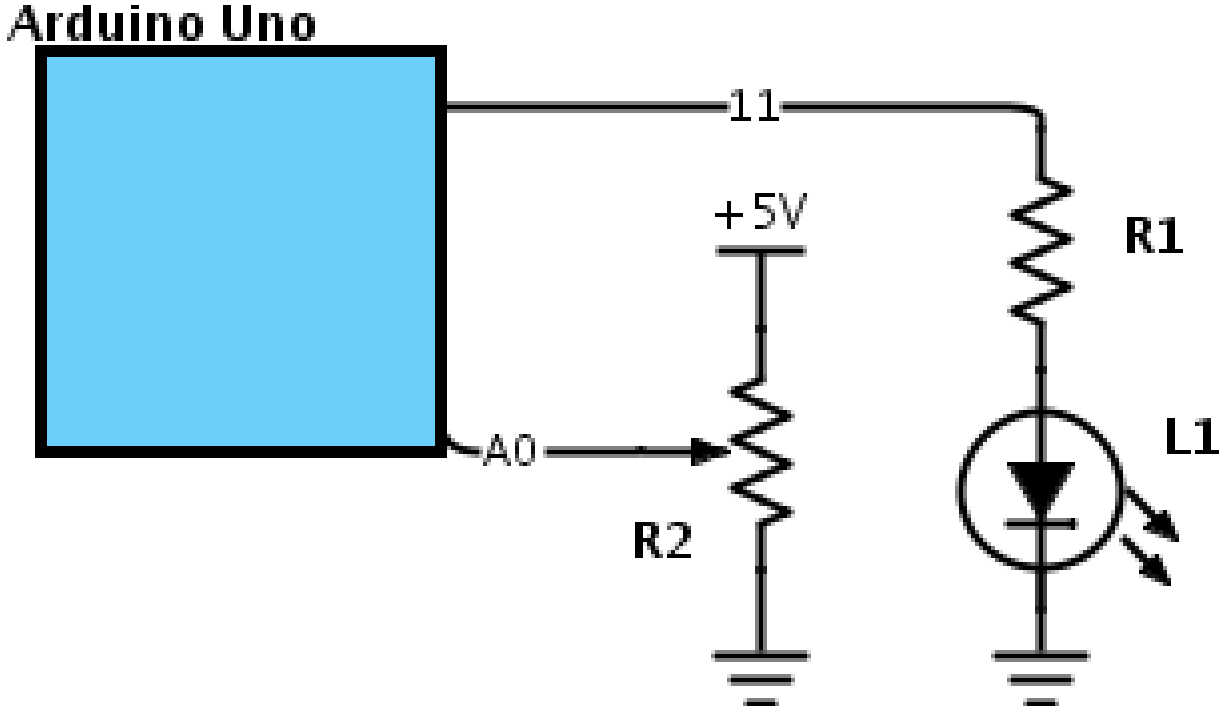

- La fonction d'initialisation `setup`

```
void setup() {
  ...
  pinMode(pinS1, INPUT_PULLUP);
  ...
```

- Traitement : bouton enfoncé = front descendant

# 2c4. Contrôler la luminosité d'une LED via un potentiomètre

- Intérêt
  - Entrée et sortie analogiques
    - Il n'y a pas de sortie analogique sur notre carte : elle est simulée par une sortie numérique en mode MLI/PWM
- Montage électronique



# 2c4. Contrôler la luminosité d'une LED via un potentiomètre

- Fonction d'initialisation `setup`

```
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
}
```

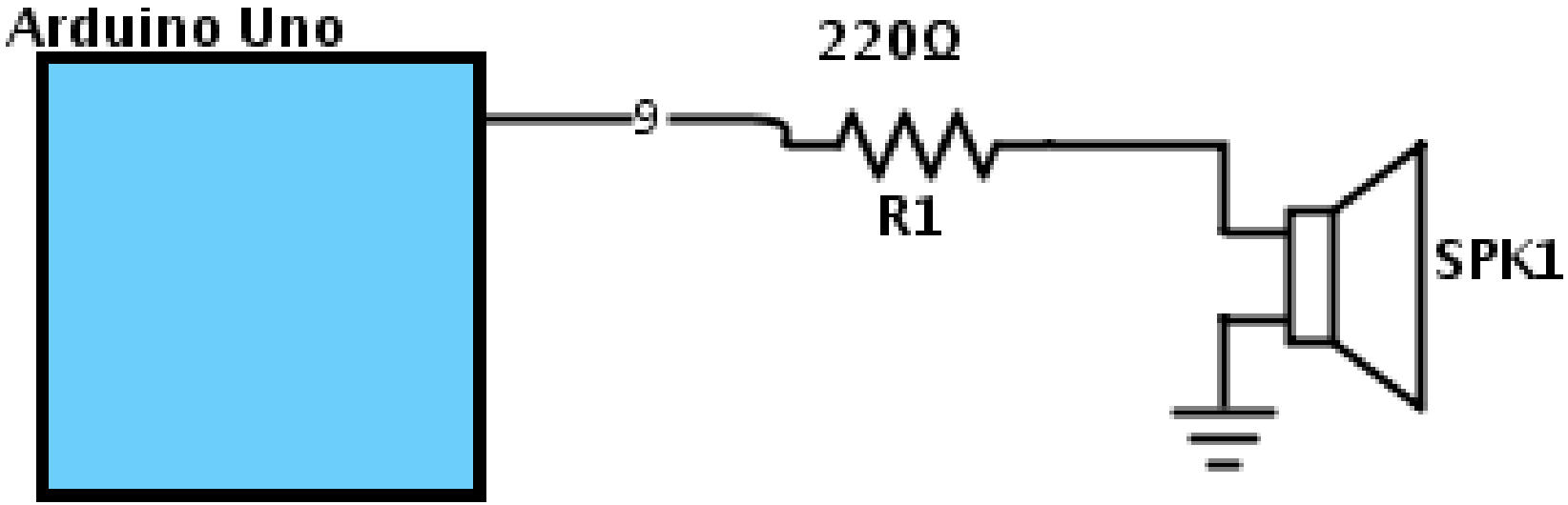
# 2c4. Contrôler la luminosité d'une LED via un potentiomètre

- Fonction de traitement `loop`

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

# 2c5. Piloter un buzzer piézoélectrique

- Intérêt
  - Fréquence de travail plus grande : quelques dizaines de kHz (monde des µs)
- Montage électronique



# 2c5. Piloter un buzzer piézoélectrique

- Fonction d'initialisation `setup`

```
const int brocheBuzzer = 9; // 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
}
```

# 2c5. Piloter un buzzer piézoélectrique

- Fonction de traitement `loop`

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

# 2c6. Etude de cas : une voiture de police

- objectif : conception d'un jouet type *voiture de police* disposant d'un maximum de gadgets
  - Les phares (avant et arrière) sont sensibles à la lumière (relation inverse)
  - Un gyrophare et une sirène (avec les fréquences normalisées de la Police Nationale) activables au besoin
  - Clignotants activables
  - Les roues arrière sont motorisées : marche avant, marche arrière, réglage de la vitesse
  - Système de freinage : allumage des feux arrières quand moteur en décélération
  - Les roues avant sont orientables : direction asservie à un potentiomètre

# Plan

1. Présentation générale
   a) Les microcontrôleurs
   b) Leur programmation
2. Les cartes Arduino
   a) Le matériel
   b) L'environnement de programmation
   c) Quelques montages, avec rappels d'électronique
3. Conclusion – perspectives

Systèmes concurrents

# Conclusion

- Avantages d'un kit Arduino :
  - Facile à mettre en œuvre
  - Faible coût
  - Grande richesse de cartes d'extensions ou de composants (capteurs, actionneurs)
- Inconvénients :
  - Pour un informaticien de formation : il faut avoir quelques connaissances en électronique (analogique et logique)
  - Il est possible de griller le contrôleur USB de son PC, voire sa carte mère ☹

# Les autres solutions similaires

- Les solutions type microcontrôleurs :
  - Famille AVR (Atmel)
    - Compatibles Arduino : Freeduino, Boarduino, …
  - Famille ARM
  - Famille PIC (MicroChips)
  - Famille LPC (NXP [ex-Philips], sous licence Intel)
  - Circuits spécialisés type FPGA ou PSoC
  - Presque tous les fabricants de processeurs conçoivent des microcontrôleurs :
    - Texas Instrument, FreeScale (ex Motorola), …

# Les autres solutions similaires

- Les solutions type nano-PC :
  - Principe : vrai PC (processeur 32 ou 64 bits) tout en un de petite taille, avec système d'exploitation multitâches (Linux, Androïd, …)
- Quelques exemples :
  - Raspberry Pi
  - CI20 (Imagination)
  - Lemel (PC Stick, Intel)
  - Wyse Cloud Connect (Dell)
  - ChromeCast (Google)
  - …

# A propos des aspects électroniques

- Magasins d'électronique (souvent en VPC)
  - Lextronic
  - Selectronic
- Logiciels liés à l'électronique
  - SchemeIt (`digikey.com`) : éditeur de schémas
  - Fritzing (`fritzing.org`) : éditeur de schémas + montage sur platine d'essai + circuit imprimé (PCB)

# Bibliographie

- Ouvrages à la BIU/S, dont :
  - **BARTMANN Erik** : *Le Grand Livre d'Arduino* ; éditions Eyrolles, collection Serial Markers, 2014. ISBN 978-2-212-13701-9.
- Sites internet :
  - `http://arduino.cc`
    - Site de référence : spécifications des cartes, environnement de développement C/C++
  - Wikipedia
  - Perso :

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