# Microcontrôleurs pour les sciences physiques

**David THERINCOURT**
**Lycée Roland Garros - Académie de la Réunion**
21 mai 2025

*Plus d'informations sur* https://physique-microcontroleurs.readthedocs.io/

# Table des matières

Les nouveaux programmes 2019 en seconde générale et technologique, en première générale et en terminale générale introduisent l'utilisation des **microcontrôleurs** et des **capteurs** en sciences physiques.

Cette document a pour objectif d'apporter aux professeurs de physique-chimie les notions techniques nécessaires sur les microcontrôleurs afin d'aborder au mieux les nouvelles **capacités numériques**.

# Chapitre 1

# Introduction

## 1.1 Qu'est-ce qu'un microcontrôleur ?

Un microcontrôleur est un circuit intégré regroupant un micro-processeur, de la mémoire et des périphériques sur la même puce. Contrairement à un microprocesseur classique, un **microcontrôleur est surtout utilisé pour une application électronique spécifique**.

De nos jours, les microcontrôleurs sont présents un peu partout : dans les appareils domestiques, médicaux, de télécommunication, dans les voitures, les avions, l'industrie, ...

Apparus dans les années 70, les microcontrôleurs à architecture 8 bits ne sont pas près de disparaître. Très peu chère, on les retrouve dans des petites applications (ex. télécommande). La célèbre carte Arduino UNO fonctionne avec un microcontrôleur 8 bits !

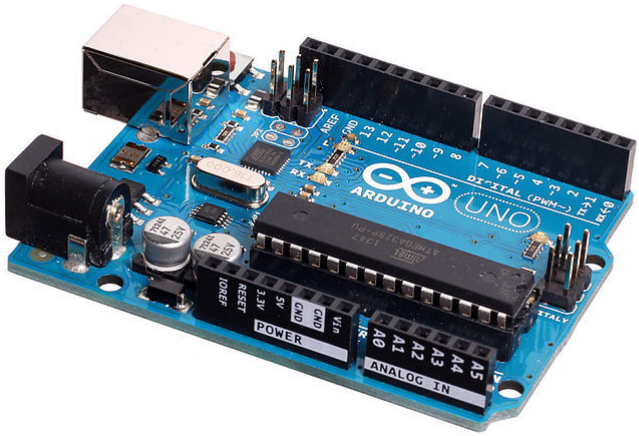



FIG. 1 – Carte Arduino UNO (microcontrôleur Atmel ATMEGA 328)

Actuellement, la tendance est aux microcontrôleurs 32 bits (ex. ARM Cortex-M, STM32, ...) qui sont plus adaptés aux applications plus évoluées. C'est ce type de microcontrôleur qui a permis le portage du langage Python (MicroPython) au sein des microcontrôleurs. Les cartes Micro :bit, Pyboard ou encore à base d'ESP32 en sont les parfaits exemples !

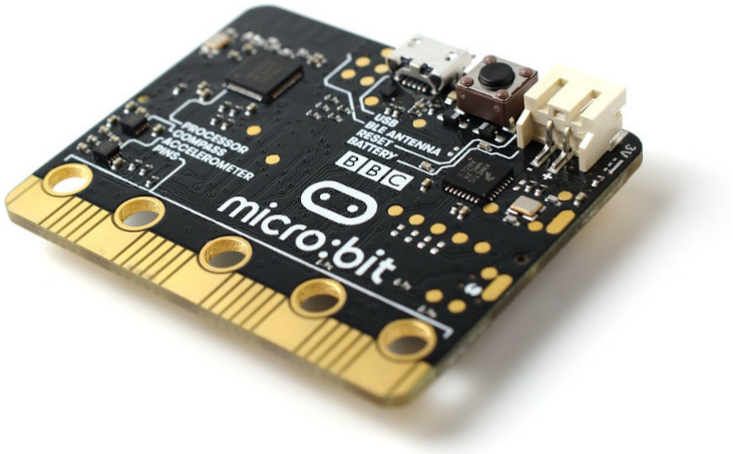



FIG. 2 – Carte micro :bit

FIG. 3 – Carte PyBoard (microcontrôleur STM32)

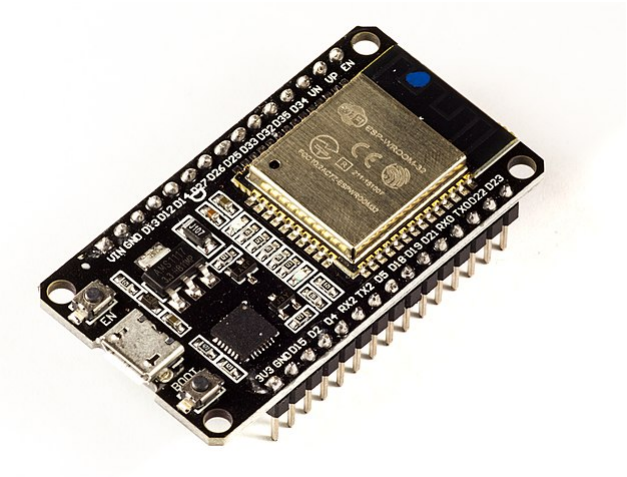


FIG. 4 – Carte ESP-WROOM-32 (microcontrôleur ESP32)

## 1.2 Pourquoi des microcontrôleurs en sciences physiques ?

Le monde actuel est fortement imprégné par le numérique. Par exemple, les téléphones portables et les objets connectés comportent une **multitude de capteurs** mesurant des grandeurs très variées comme la température, la fréquence cardiaque, la pression, l'accélération, les ondes sonores, ...

Il est donc important d'expliquer comment s'effectue la **mesure d'une grandeur physique analogique** un appareil numérique.

Il en est de même pour la **génération de signaux** (ex. son) à partir d'un appareil numérique.

## 1.3 Les fonctions d'un microcontrôleur

Les microcontrôleurs permettent principalement de :

- **générer de signaux** (ex. son, impulsion de commande, ...).
- **mesurer des tensions** (ex. adaptation de capteurs analogiques, acquisition de signaux, ...).
- **mesurer des durées** (ex. période, fréquence, temps caractéristique, ...).

## 1.4 Applications en sciences physiques

De manière générale, les microcontrôleurs sont utilisés :

- pour réaliser des **petites applications** (ex. thermomètre, télémètre à ultrasons, ...) en rapport avec un cours ou un TP ;
- dans des **projets** (enseignement scientifique).

Avec des capteurs, il est en plus possible de :

- réaliser des **mesures** (ex. température, célérité son, pression, ...) ;
- faire de **l'acquisition de données** en mode **autonome** (ex. mesure de pression sur un ballon sonde) ou mode **connecté** (branché à un ordinateur).

# Chapitre 2

# Microcontrôleurs

## 2.1 Cartes Arduino

### 2.1.1 Qu'est-ce qu'Arduino ?

**Arduino** est une **carte électronique** à base de microcontrôleur (ex. ATMEL AVR ATMEGA) développée par Arduino.cc sous licence libre. Tous les schémas des cartes sont disponibles librement sur le Web. A peu près une vingtaine de versions de cartes officielles ont été fabriquées dont la célèbre l'Arduino UNO.

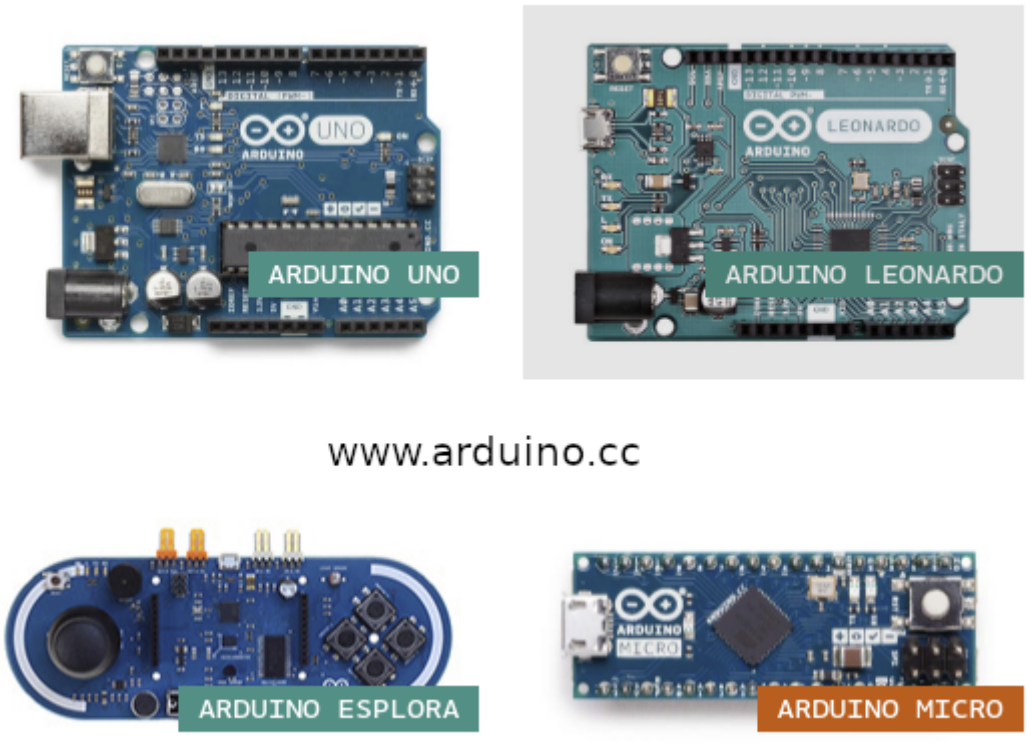



FIG. 1 – image : www.arduino.cc

A l'origine conçue pour la **création artistique**, la carte Arduino trouve des applications dans des domaines d'applications aussi variés qu'insolites. Une carte Arduino s'utilise généralement comme :

— **dispositif autonome** dans des applications comme la domotique, la robotique, les systèmes embarqués, ...

— **interface** entre un ordinateur (logiciel tiers) et des capteurs ou des actionneurs ;

**Arduino** est aussi le **logiciel de développement** des cartes du même nom. Également sous licence libre, cet environnement de développement intégré (IDE) utilise C/C++ comme langage de programmation et le port USB pour le téléversement du programme obtenu.

### 2.1.2 La carte Arduino UNO (Rev 3)

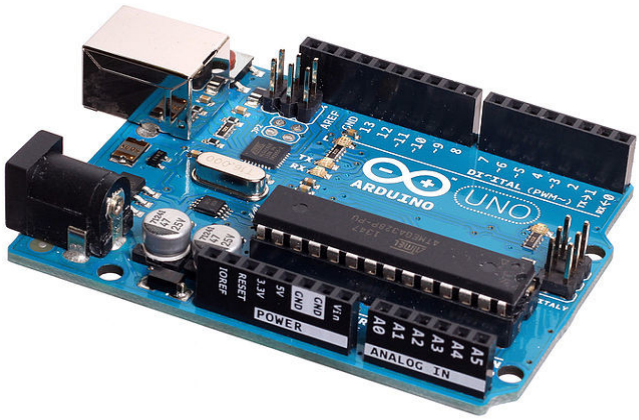



FIG. 2 – Arduino Uno R3

**Arduino UNO** est une des cartes officielles les plus récentes et économiques.

Caractéristiques principales :

— microcontrôleur 8 bits ATMEGA328P cadencé à 16 Mhz ;
— alimentation externe (7 à 12 V) ou USB (5 V) ;
— programmation et communication via port USB ;
— 14 broches d'entrées/sorties numériques dont 6 PWM ;
— 6 entrées analogiques sur 10 bits ;
— 1 port I2C (communication avec capteurs/actionneurs numériques) ;
— 1 port UART (communication série) ;
— 3 timers (comptage et mesure de temps) ;
— gestion des interruptions.

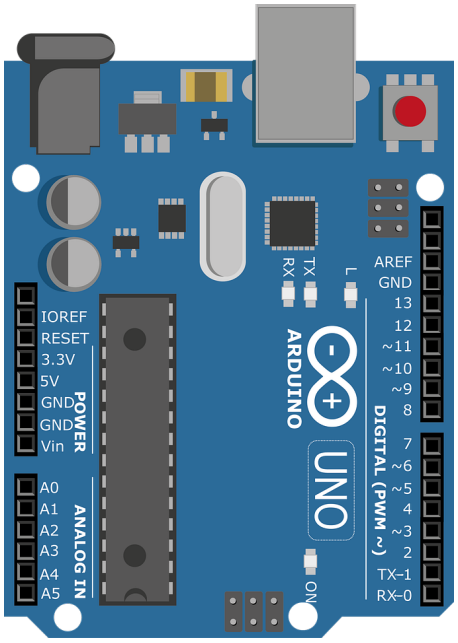



FIG. 3 – Brochage de l'Arduino Uno R3

**Avertissement**

Les niveaux de tension acceptables sur les broches d'entrées doivent être **comprises entre 0 V et 5 V** sous peine de détruire le microcontrôleur ou la carte !

**Note**

La carte Arduino UNO ne possède pas de vraies sorties analogiques mais des sorties à **modulation de largeur d'impulsion** (MLI). Ce sont les six fameuses **sorties PWM** (Pulse Width Modulation).

### 2.1.3 Educaduino Lab (Eurosmart)

https://educaduino-lab.com/

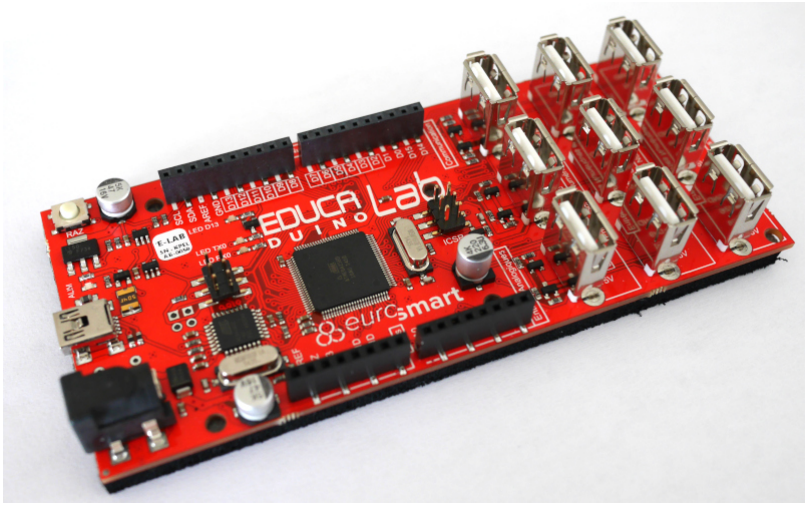



FIG. 4 – La carte Educaduino-Lab (E-LAB)

La carte **Educaduino Lab** a été conçue sur la base d'une carte Arduino MEGA 2560. Cette dernière est équivalente à une carte arduino UNO mais avec plus de mémoire et surtout **plus de ports d'entrée/sortie**. Ce qui a permis à Eurosmart d'y placer des **connecteurs USB pour ses propres capteurs** tout en gardant la connectique classique de l'Arduino UNO.

Caractéristiques principales :

— microcontrôleur ATMEGA 2560 (comme l'Arduino MEGA 2560) ;
— protection des ports d'entrée/sortie ;
— brochage compatible Arduino Uno Rev 3 (pin 0.8mm, shield Grove, ...) ;
— ports supplémentaires en USB pour capteurs Educaduino-Lab ;

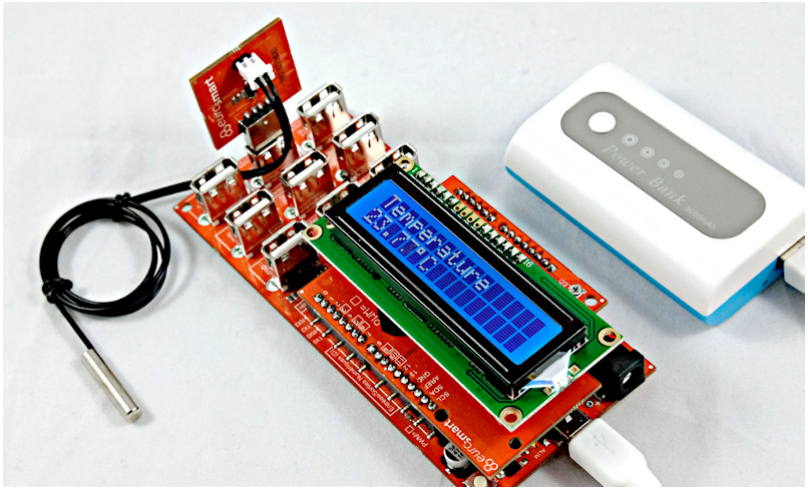



FIG. 5 – Mesure d'une température (image : Eurosmart)

Une malette avec un afficheur LCD et plusieurs capteurs adaptés au programme du lycée est également proposée.

FIG. 6 – Kit sciences-physiques 2nde/1ère (image : Eurosmart)

### 2.1.4 Plug'Uino® Uno (Sciencéthic)

https://www.sciencethic.com/

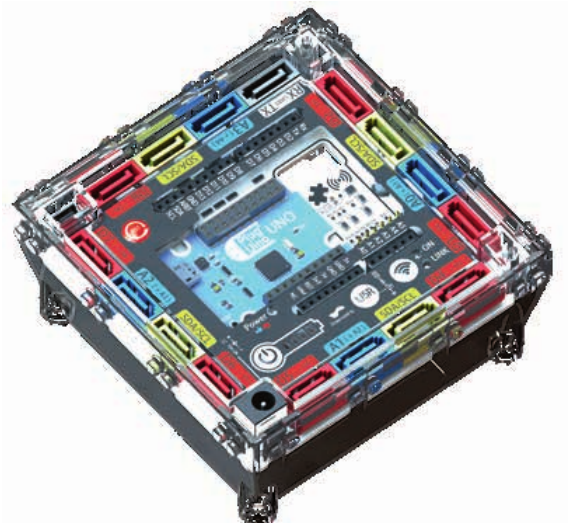


FIG. 7 – La carte Plug'Uino ® Uno (image : Sciencéthic)

Sciencéthic propose également une carte **Plug'Uino Uno** protégée contre les mauvaises manipulations et 100% compatible Arduino UNO Rev 3.

Caractéristiques principales :

— microcontrôleur ATMEGA 328P (comme l'Arduino Uno) ;

— protection des ports d'entrée/sortie ;
— brochage compatible Arduino Uno Rev 3 (pin 0.8mm, shield Grove, ...) ;
— connecteurs SATA pour les capteurs Plug'uino ;

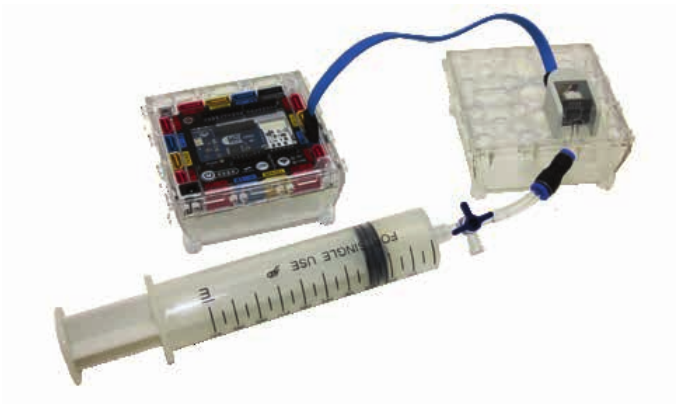


FIG. 8 – Capteur de pression et loi de Mariotte (image : Sciencéthic)

## 2.2 Cartes Pyboard

### 2.2.1 La carte officielle Pyboard

FIG. 9 – Carte Pyboard v1.1

Créée par le physicien australien Damien George, la Pyboard est la carte officielle de la distribution MicroPython. Construit autour d'un microcontrôleur STM32 de STMicroelectronics, il s'agit d'une des plus puissantes cartes de développement du marché.

**Principales caractéristiques de la Pyboard v1.1 :**

— Microcontrôleur STM32F405 (ARM Cortex M4 - 32 bit - 168 Mhz) ;

— 1024 ko de mémoire flash (ROM) ;* 192 ko de mémoire vive (RAM) ;
— niveau de tension à 3,3V ;
— 1 port micro USB (programmation REPL + accès mémoire flash) ;
— lecteur micro SD ;
— 2 boutons, 4 LED, 1 accéléromètre 3 axes ;
— 35 ports E/S ;
— 3 CAN disponibles sur 16 broches ;
— 2 CNA disponibles sur 2 broches (X5 et X6) ;
— 17 timers.

**Brochage :**

Les broches d'entrée/sortie sont réparties en deux blocs : X1 à X12 et Y1 à Y12.

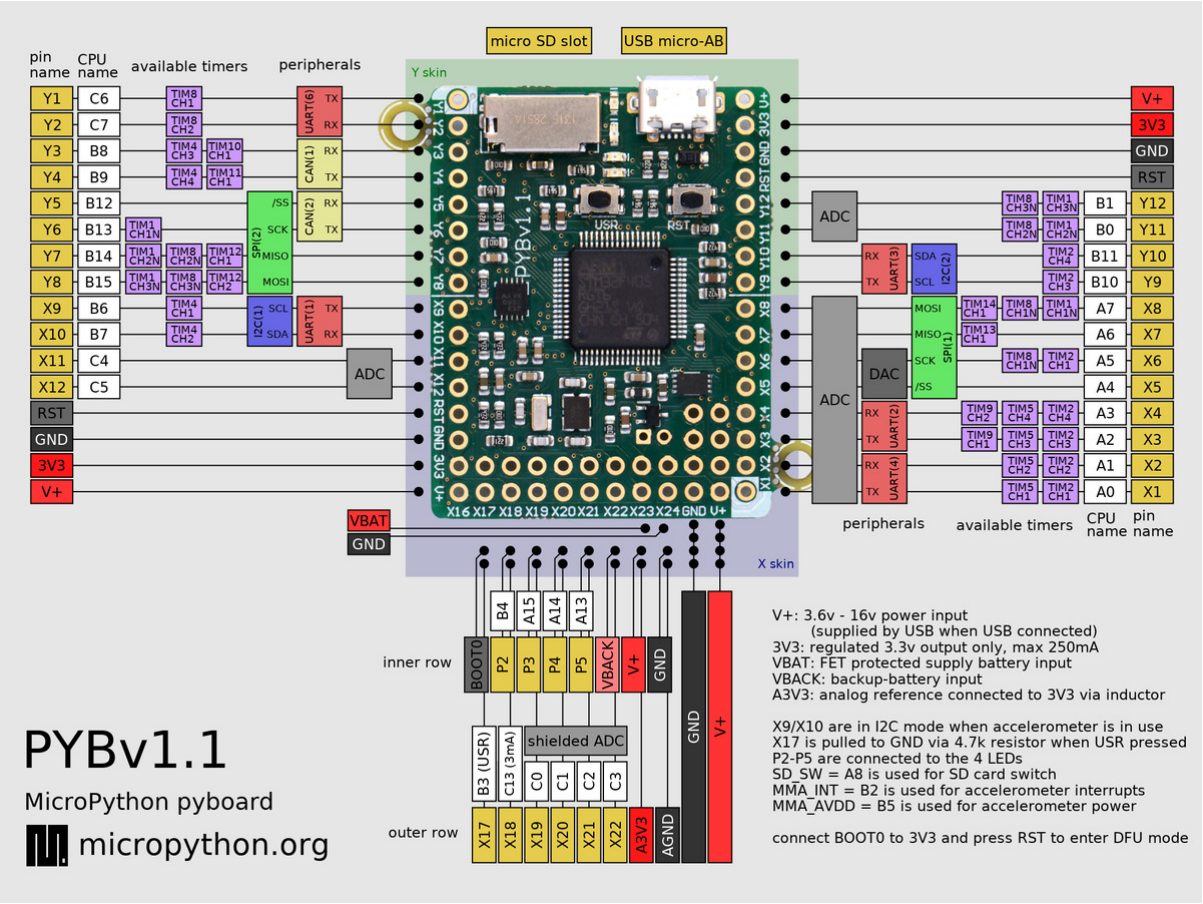



FIG. 10 – Image : micorpython.org

### 2.2.2 La carte Plug'Uino PY de Sciencethic

Le fournisseur Sciencethic propose une carte Pyboard (STM32F405) spécialement conçue pour les sciences physiques **Plug'Uino PY**.


FIG. 11 – Image : Sciencethic

Par rapport à la carte officielle, cette carte a l'avantage de disposer :

— de **protections des entrées/sorties** contre les mauvaises manipulations ;
— d'une **adaptation du niveau des tensions à 5V** en logique et en analogique ;
— d'un brochage Arduino en plus des connecteurs SATA pour les capteurs.

Elle dispose en autres :

— d'un écran OLED 64x32 pixels ;
— d'un accéléromètre 3D ;
— d'un lecteur de carte SD.

### 2.2.3 La carte Feather STM32F405 Express d'Adafruit

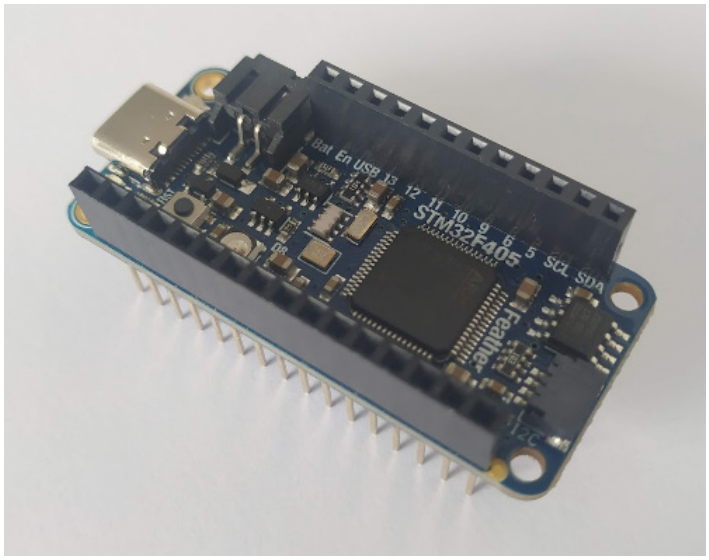



FIG. 12 – Carte Feather STM32F405 Express

Dans la famille Feather d'Adafruit, la carte de développement Feather STM32F405 Express est équivalent à la Pyboard. Associée au Grove Shield Featherwing , cela peut-être une alternative intéressante pour les sciences physiques.

#### 2.2.3.1 Précautions

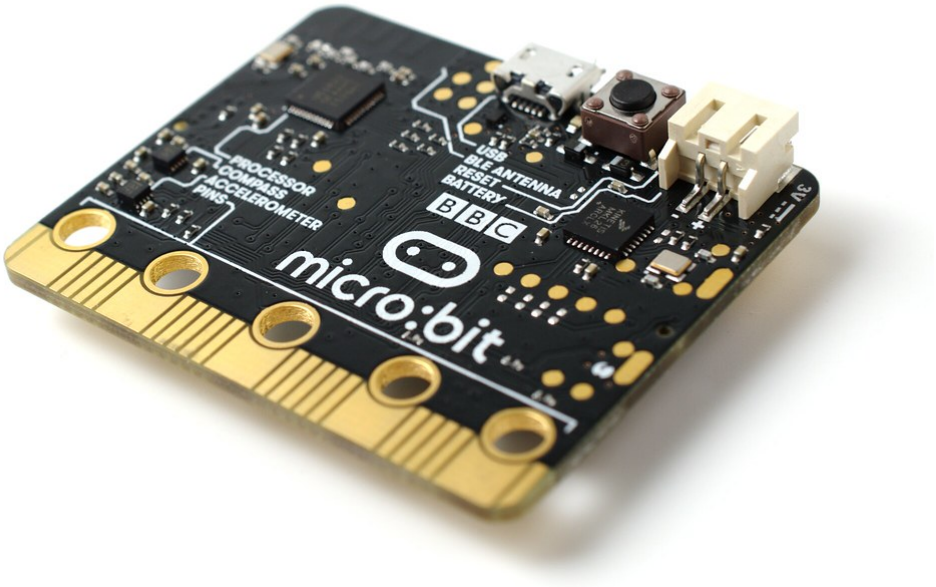
**Attention**

Les cartes Pyboard sont alimentées sous 3,3 V. Même si les microcontrôleur STM32F405 ont une **tolérance de 5 V sur les entrées logiques**, ce n'est pas le cas pour **les tenions appliquées sur les entrées analogiques qui ne doivent pas dépassées les 3,3 V** sous peine de détruire la carte.

Il faudra donc bien faire attention à la compatibilité des niveaux de tension lors du choix des capteurs (ex. capteur de pression). Les capteurs actifs alimentés sous 5 V ne fonctionneront pas !

Ces précautions ne s'appliquent pas pour la carte **Plug'uino Py** car ce dernière est compatible 5 V.

## 2.3 Cartes Micro :bit



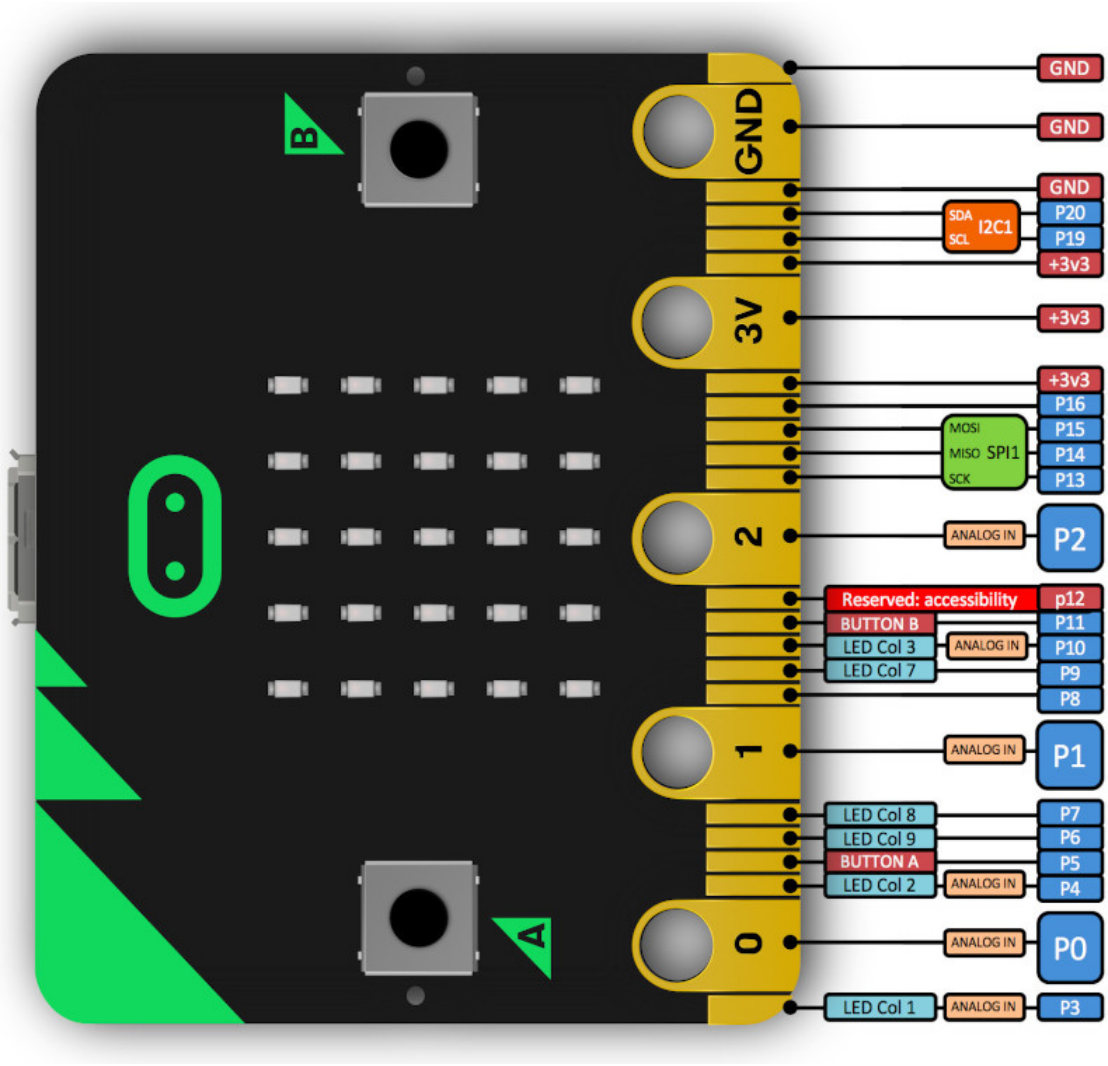

La carte Micro :bit a été développée par la BBC pour l'enseignement de l'informatique dans les écoles du Royaume Uni. Elle est programmable en MakeCode ou en MicroPython.

Principales caractéristiques de la carte :

- Microcontrôleur nRF51822 (ARM Cortex M0 - 32 bit - 16 Mhz) ;
- 256 ko de mémoire flash (ROM) ;
- 16 ko de mémoire vive (RAM) ;
- Tension de fonctionnement à 3,3V ;
- 1 port micro USB (programmation REPL + accès mémoire flash) ;
- 25 ports E/S ;
- 3 entrées analogique ;
- 2 boutons, matrice 5x5 Leds, 1 ;
- Bluetooth 4.0 LE
- Magnétomètre 3D, accéléromètre 3D

Brochage :

Le brochage est assez particulier. Cinq anneaux (grosses broches) donnent accès à 3 entrées/sorties (P0, P1 et P2) et à l'alimentation. Les autres ports sont accessibles sur des petites broches.


B
A
GND
3V
2
1
0
GND
GND
GND
SDA
SCL
I2C1
P20
P19
+3v3
+3v3
+3v3
P16
MOSI
MISO
SCK
SPI1
P15
P14
P13
ANALOG IN
P2
Reserved: accessibility
p12
BUTTON B
P11
LED Col 3
ANALOG IN
P10
LED Col 7
P9
P8
ANALOG IN
P1
LED Col 8
P7
LED Col 9
P6
BUTTON A
P5
LED Col 2
ANALOG IN
P4
ANALOG IN
P0
LED Col 1
ANALOG IN
P3

# Chapitre 3

# Langage Arduino

## 3.1 Logiciel Arduino IDE

### 3.1.1 Arduino IDE (version 1.8)

Le logiciel **Arduino IDE 1.1** est un environnement intégré de développement (IDE) multiplaforme. Il est téléchargeable sur le site officiel http://www.arduino.cc/en/

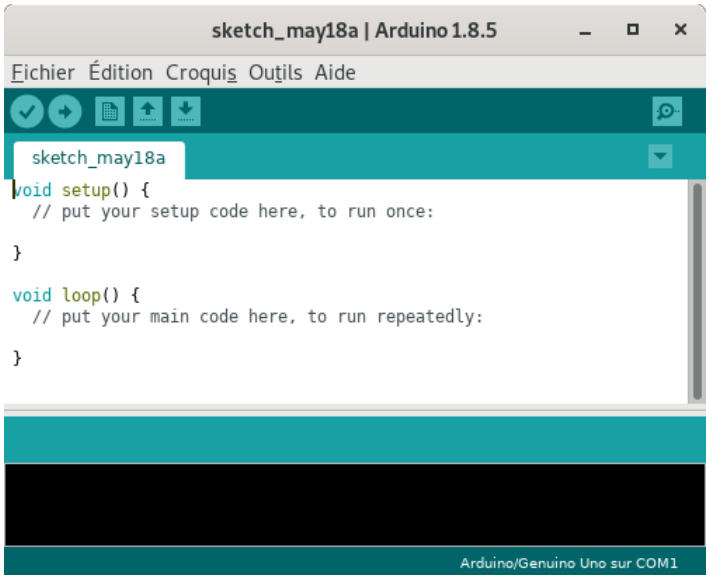



FIG. 1 – Logiciel Arduino IDE 1.8

### 3.1.2 Arduino IDE (version 2.x)

Arduino IDE 1.1 (devenu obsolète) a été remplacé par Arduino IDE 2 avec une interface plus moderne !


FIG. 2 – Logiciel Arduino IDE 2

### 3.1.3 Etapes de la mise en oeuvre d'un projet Arduino

La mise en œuvre d'un projet Arduino s'effectue dans l'ordre suivant :

1. **Édition** du programme dans l'éditeur de l'interface ;
2. **Vérification** du programme (compilation) ;
3. **Téléversement** du programme sur la carte Arduino ;
4. **Exécution** du programme sur le carte Arduino (de façon autonome sans ordinateur).

## 3.2 Premier programme : Blink

Le programme **Blink** propose de faire clignoter la LED intégrée à la carte de développement. Cette LED est connectée en interne à la broche 13.

### 3.2.1 Edition

Le programme **Blink** est disponible dans les exemples du logiciel **Arduino IDE**.

Dans le menu `Fichier > Exemples > Basics > Blink`.

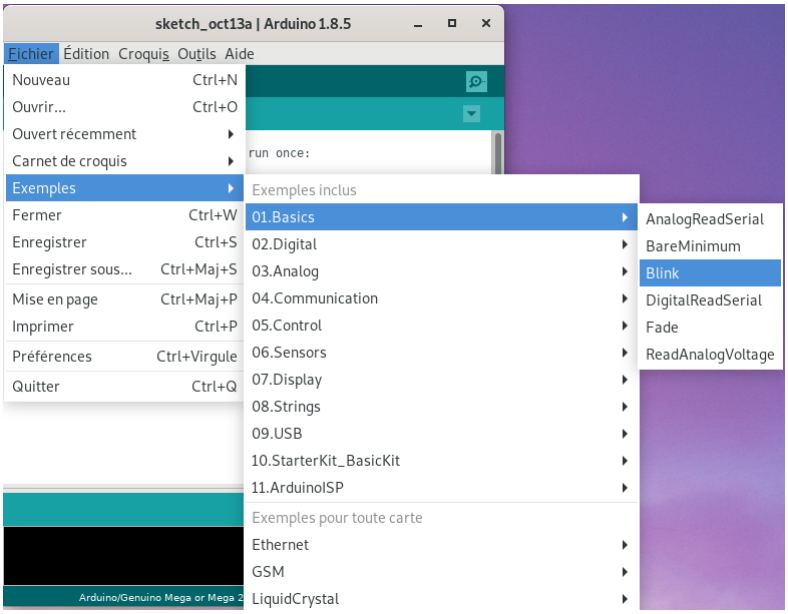



FIG. 3 – Ouvrir le programme Blink


FIG. 4 – Edition du programme Blink

**Note**

— Un programme Arduino écrit en **langage C/C++** est composé **d'une suite d'intructions**.
— Ces instructions sont exécutées dans **l'ordre des lignes de code**.
— Les **commentaires** en gris sont délimités par les caractères `/*` et `*/` sur plusieurs lignes ou commencent pas les caractères `//` sur une même ligne.

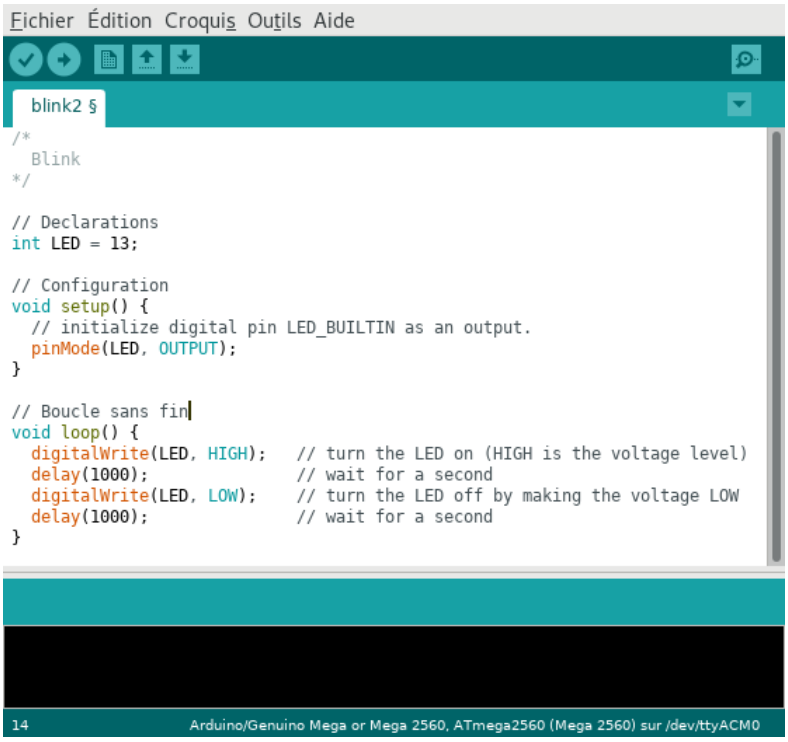



FIG. 5 – Une version modifiée du programme Blink

**Avertissement**

Un progamme Arduino respecte toujours une **structure spécifique** composée en trois parties :

- Les déclarations : **définitions** des constantes et des variables ;
- La fonction `setup()` : **configuration** de la carte (entrées, sorties, port série, ...) ;
- La fonction `loop()` : **instructions du programme exécutées** dans une **boucle infinie** (sans fin).

### 3.2.2 Compilation

**Avertissement**

Avant de lancer la compilation, il est important de **choisir le modéle de carte Arduino utilisé**. Le programme généré est dépendant du type de microcontrôleur présent sur la carte.

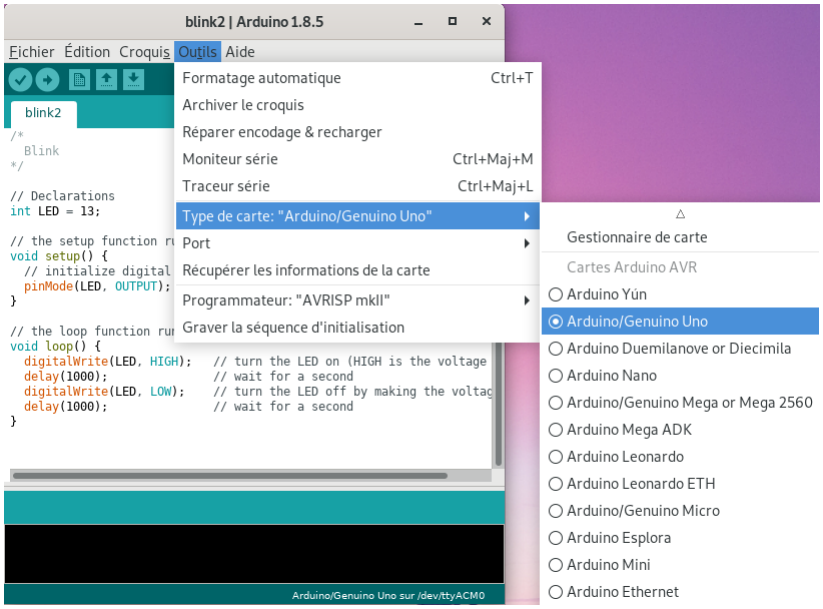



FIG. 6 – Choix du type de carte

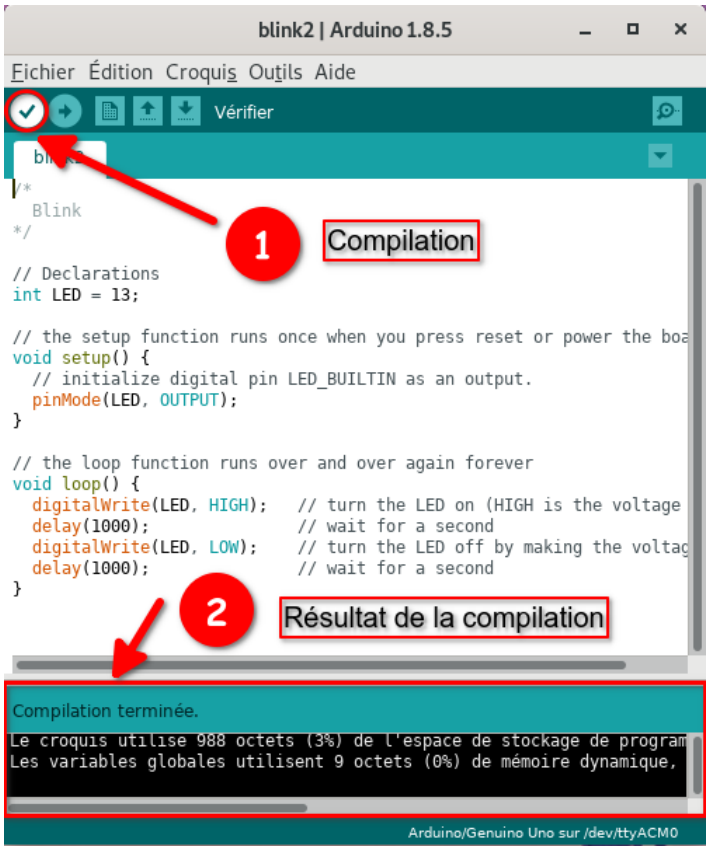



FIG. 7 – Puis la compilation peut s'effectuée !

### 3.2.3 Téléversement

**Avertissement**

Pour téléverser le programme obtenu, il est nécessaire de **sélectionner le port de communication série** sur lequel est connectée la carte Arduino.

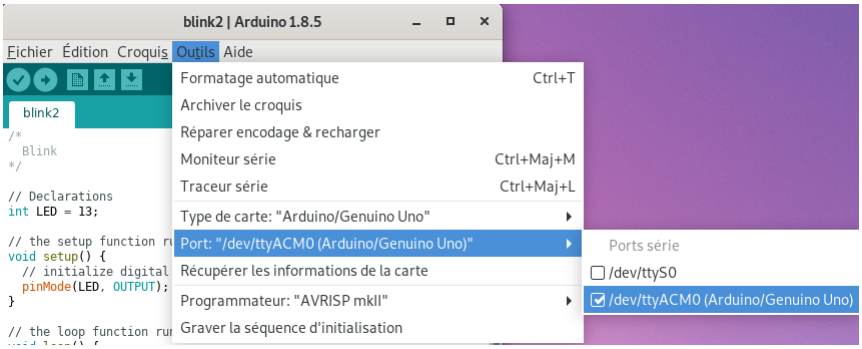



FIG. 8 – Choix du port de communication


FIG. 9 – Téléversement du programme

### 3.2.4 Exécution

Le programme s'exécute sur la carte Arduino de façon autonome (sans ordinateur).

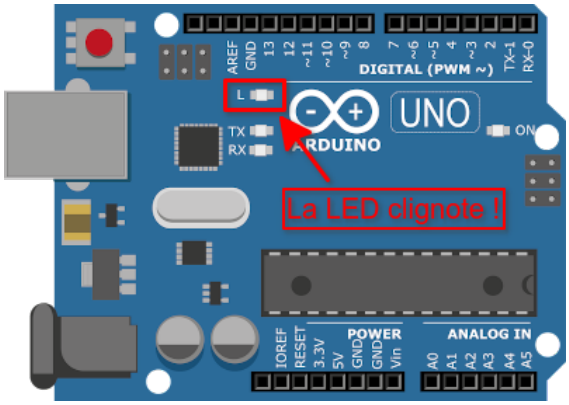



FIG. 10 – Exécution du programme Blink sur la carte Arduino Uno R3

## 3.3 Spécificités du langage Arduino

Le langage de programmation C/C++ est utilisé par le logiciel Arduino pour programmer les microcontrôleurs Arduino.

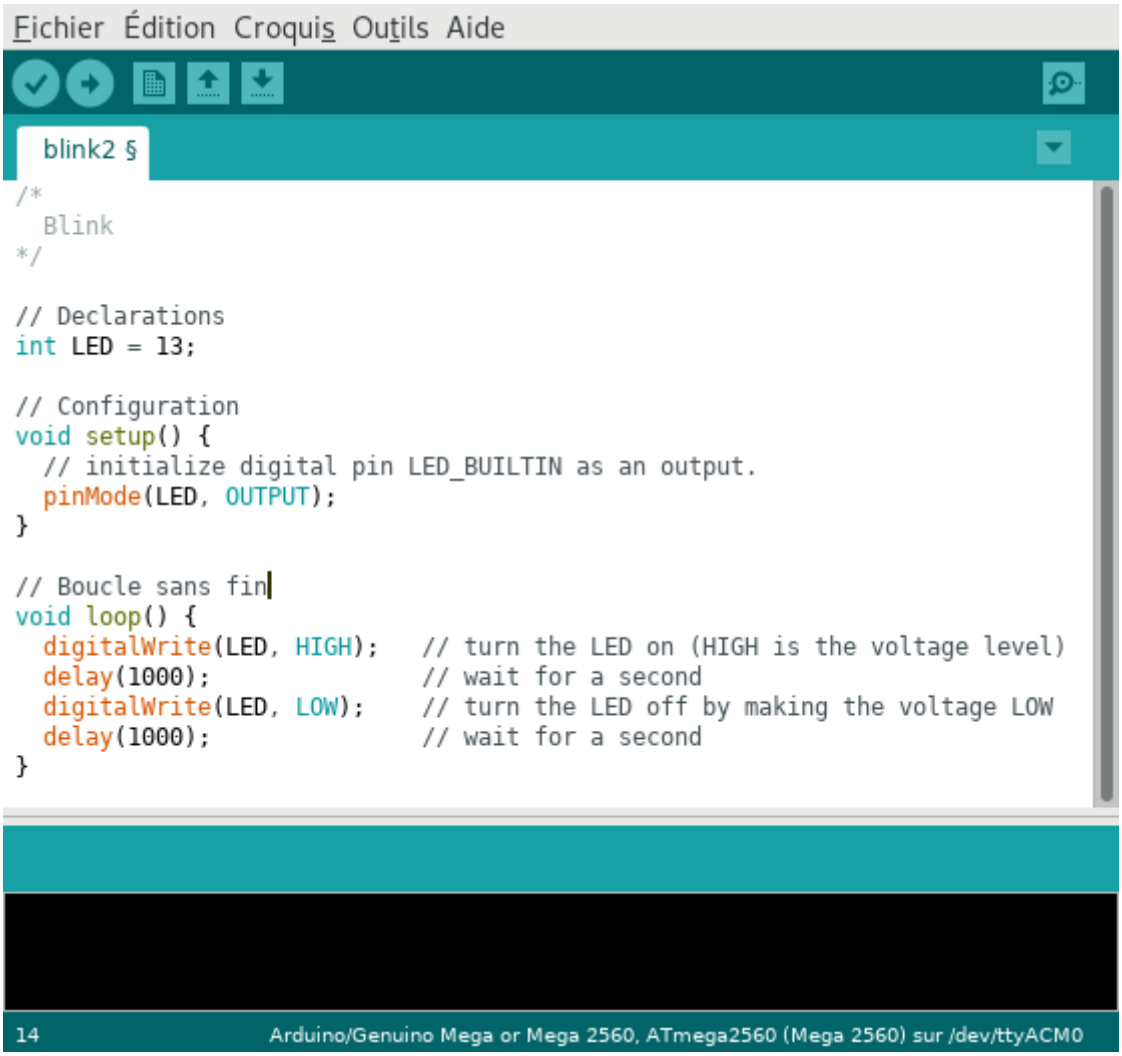



FIG. 11 – Edition d'un programme Arduino

### 3.3.1 Syntaxe

— Toutes les instructions se terminent par un point virgule `;` sauf pour les directives `#include` et `#define`.
— Les blocs d'instructions sont délimités par des accolades `{...}`.
— Les **commentaires** en gris sont délimités par les caractères `/*` et `*/` sur plusieurs lignes ou commencent pas les caractères `//` sur une même ligne.

#### 3.3.1.1 Typage des variables

Le type d'une variable doit être renseigné à sa déclaration.

Quelques types disponibles :

| Type | Description | Valeurs |
|---|---|---|
| `int` | entier sur 16 bits | -32768 à 32767 |
| `long` | entier sur 32 bits | -2147483648 à 2147483647 |
| `float` | flottant sur 32 bits | -3.4028235E+38 à -3.4028235E+38 ; |
| `char` | caractère sur 8 bits | Table ASCII |

Exemples :

```
int a = 5;
float pi = 3.14;
char c = 'A';
```

### 3.3.2 Constantes prédéfinies

Afin d'améliorer la lecture du code, des constantes sont définies.

| Constante | Valeur |
|---|---|
| `LOW` | 0 (niveau logique) |
| `HIGH` | 1 (niveau logique) |
| `OUTPUT` | broche en sortie |
| `INPUT` | broche en entrée |
| `LED_BUILTIN` | 13 (numéro de broche de la LED intégrée) |

### 3.3.3 Structure du programme

Un programme Arduino respecte toujours une **structure spécifique** composée en trois parties :

— Les déclarations : **définitions** des constantes et des variables ;
— La fonction `setup()` : **configuration** de la carte (entrées, sorties, port série, ...) ;
— La fonction `loop()` : **instructions du programme exécutées** dans une **boucle infinie** (sans fin).

### 3.3.4 Documentation

Une référence complète du langage Arduino est disponible sur la site officiel d'Arduino ici.

# Chapitre 4

# Langage Python

## 4.1 Langage Micropython


FIG. 1 – Image : Sciencethic

### 4.1.1 Edition

Pour de programmer la PyBoard en langage Python natif, un micrologiciel (firmware) contenant l'interpréteur MicroPython est installé dans la mémoire flash de la carte. Ce firmware peut-être mis à jour à partir d'une version plus récente disponible sur le site de MicroPython.

Les logiciels Thonny et uPyCraft exploitent pleinement la programmation des microcontrôleurs en MicorPython. Ils permettent deux types de programmation :

— directement dans l'interpréteur MicroPython (REPL) pour tester des instructions par exemple ;

— ou dans un script (fichier avec l'extension `.py`) qui peut-être sauvegarder sur la mémoire de la carte pour être exécuté de façon autonome sur la carte.

**Avertissement**

À chaque démarrage de la carte ou lors d'un reset, les fichiers `setup.py` puis `main.py` sont exécutés dans l'ordre. C'est dans le fichier `main.py` que doit être écrit le programme !

### 4.1.2 Librairie

C'est la librairie pyb qui rassemble les fonctions spécifiques à la Pyboard.

### 4.1.3 Écrire sur une sortie digitale

Les broches de X1 à X12 et de Y1 à Y12 sont utilisables comme entrées/sorties digitales.

L'exemple suivant met la broche X1 en sortie et à l'état bas.

```
>>> from pyb import Pin             # importation de Pin dans la librairie pyb
>>> pinX1 = Pin('X1',Pin.OUT)       # La broche X1 en sortie
>>> pinX1.low()                     # LED éteinte
```

Puis à l'état haut.

```
>>> pinX1.high()
```

**Note**

Quatre LED internes sont également prises en charge par la librairie pyb.

```
>>> from pyb import LED
>>> led = LED(1)          # 1=rouge, 2=verte, 3=jaune, 4=bleu
>>> led.toggle()          # Permutation de l'état
>>> led.on()              # Allume la LED
>>> led.off()             # Éteint la LED
```

### 4.1.4 Lire une entrée digitale

L'exemple suivant met la broche X2 en entrée puis lit et affiche son état.

```
from pyb import Pin          # importation de Pin de la librairie pyb
pinX2 = Pin('X2',Pin.IN)     # La broche X2 en entrée
val = pinX2.value()          # Lecture de l'état de la broche
print(val)                   # Affichage
```

**Note**

La carte incorpore un bouton utilisateur USR connectée sur la broche X17 avec une résistance de tirage vers le haut. L'appui sur ce bouton donne donc un niveau bas !

```
from pyb import Switch
btn = Switch()
val = btn.value()        # Retourne l'état courant du bouton USR
print(val)               # Affiche la valeur (True ou False)
```

### 4.1.5 Mesurer une tension (CAN)

La conversion analogique numérique sur 12 bits est disponible sur les broches X1 à X8, X11, X12, X19 à X22, Y11 et Y12.

L'exemple suivant lit une tension sur l'entrée X19.

```
from pyb import Pin, ADC
can = ADC(Pin('X19'))   # CAN sur la broche X19
N = can.read()          # lecture d'un entier de 0 à 4095
print(N*3.3/4095)       # Affichage de la tension
```

### 4.1.6 Générer une tension analogique (CNA)

La carte Pyboard intègre deux convertisseurs numérique-analogique respectivement sur les broches X5 et X6. L'exemple suivant

L'exemple suivant applique une tension de 1,55 V sur la broche X5.

```
from pyb import Pin, DAC
cna = DAC(Pin('X5'))     # CNA sur X5
cna.write(120)           # Ecriture de 120*3,3/255 = 1,55 V
```

Les CNA définis par défaut sur 8 bits peuvent être paramétrés sur 12 bits.

```
from pyb import Pin, DAC
cna = DAC(Pin('X5'))     # CNA sur X5
cna.init(bits=12)        # Paramétrage sur 12 bit
cna.write(1500)          # Ecriture de 1500*3,3/4095 = 1,21 V
```

**Note**

Il est aussi possible de générer une tension périodique de forme quelconques.

### 4.1.7 Générer une tension MLI (PWM)

La génération d'une tension MLI est possible sur un canal d'un timer (4 canaux par timer). Voir brochage pour repérer les canaux des timers.

L'exemple suivant génère une tension MLI sur la broche X2 avec un rapport cyclique de 30%.

```
from pyb import Pin, Timer

pwm = Timer(2).channel(2, Timer.PWM, pin=Pin('X2'))  # pwm sur le canal 2 du Timer 2
pwm.pulse_width_percent(30)                          # réglage du rapport cyclique
```

**Note**

Il est intéressant ici de mesurer la tension moyenne au voltmètre numérique (entre GND et X2) en position DC.

### 4.1.8 Faire une pause

```
from time import sleep

sleep(1)  # Pause de 1 s
```

### 4.1.9 Mesurer une durée

Il est possible de mesurer la durée d'une impulsion à l'état haut ou l'état bas avec la fonction time_pulse_us() du module `machine` commun à tous les microcontrôleurs sous MicroPython.

L'exemple suivant mesure la durée (en microsecondes) à l'état haut d'une impulsion sur l'entrée X1.

```
from pyb import Pin
from machine import time_pulse_us

duree = time_pulse_us(Pin('X1'),1)
```

## 4.2 Langage BBC MicroPython


### 4.2.1 Edition

BBC micro:bit MicroPython est une édition de MicroPython spécialement conçue pour Micro :bit.

Pour son développement, il est conseillé d'utiliser l'éditeur Mu.

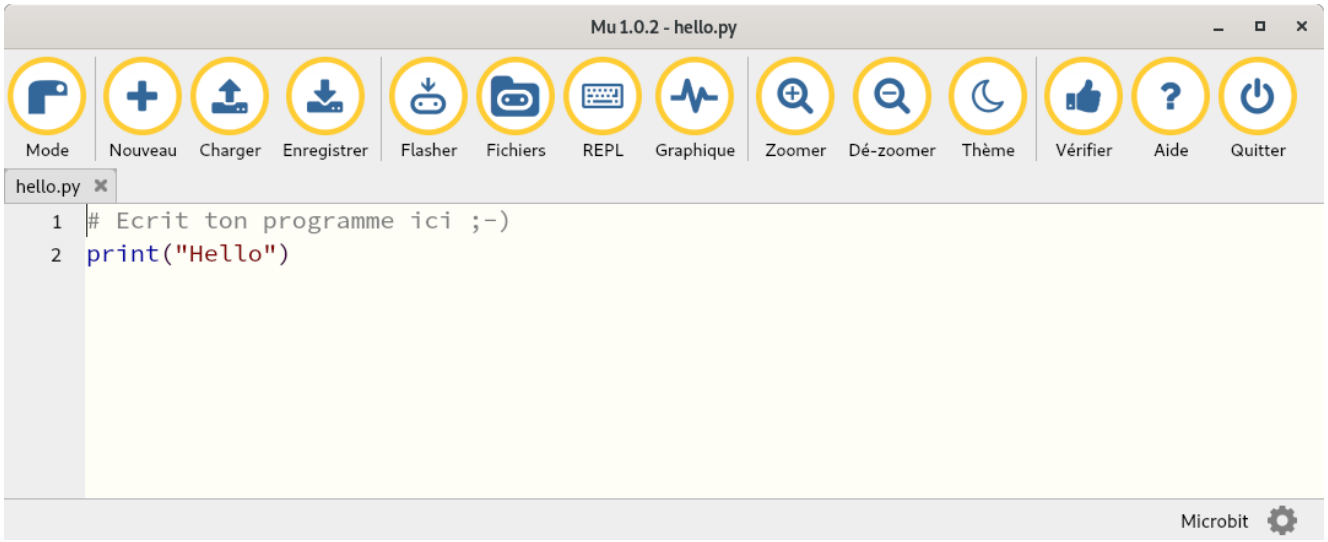



FIG. 2 – Editeur Mu

Sinon l'éditeur Thonny est la meilleure alternative à Mu.

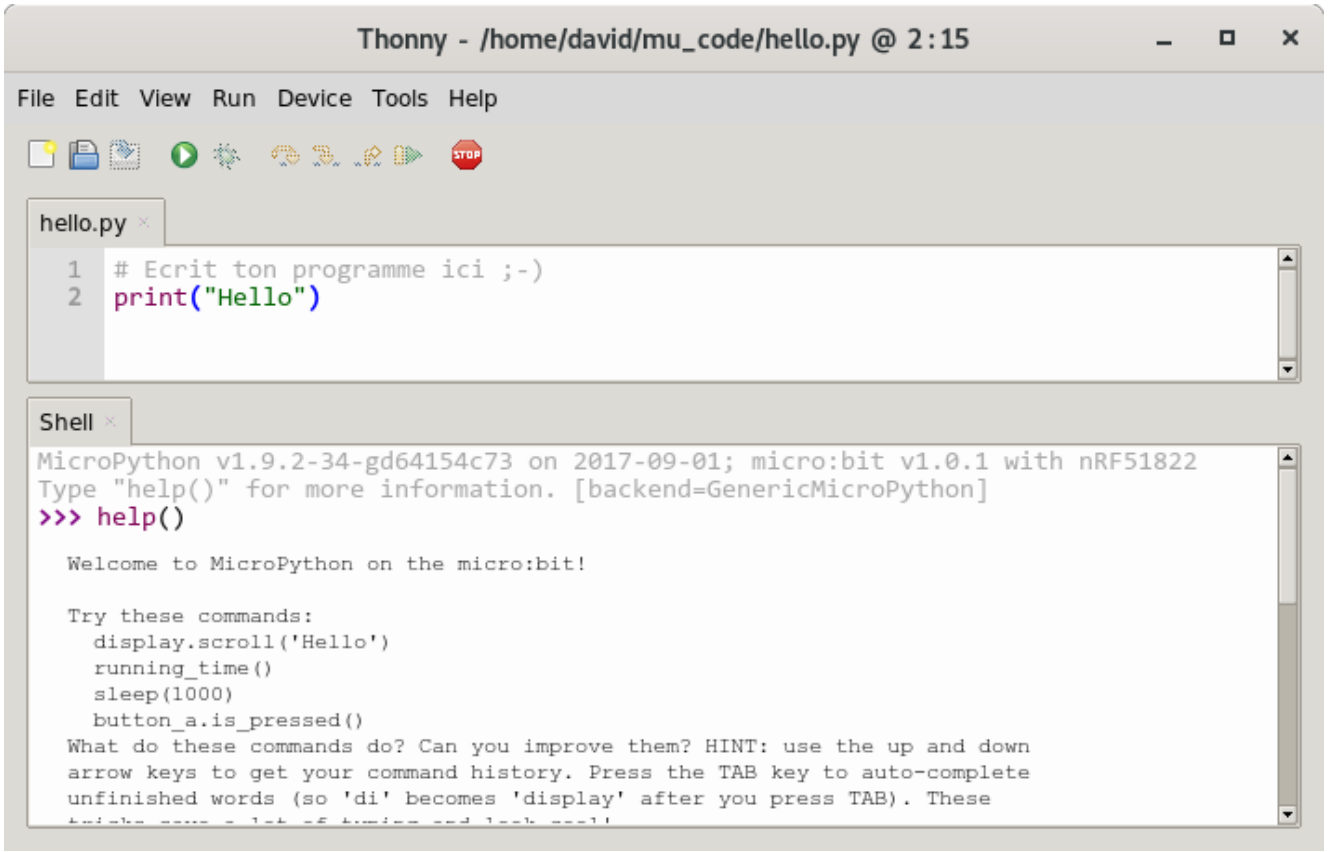



FIG. 3 – Editeur Thonny

### 4.2.2 Librairie

Les fonctionnalités spécifiques à la micro :bit sont gérées par librairie `microbit`.

Les broches sont notées sous la forme `pinN` où `N` est le numéro de la broche (ex. `pin0`, `pin`, ...).

### 4.2.3 Écrire sur une sortie digitale

La fonction `write_digital(val)` impose l'état logique `val` (0 ou 1) sur une sortie digitale.

```
from microbit import *
pin0.write_digital(1)   # Etat 1 sur P0
pin2.write_digital(0)   # Etat 0 sur P2
```

### 4.2.4 Lire une entrée digitale

La fonction `read_digital()` renvoie le niveau logique sur un broche.

```
from microbit import *
val = pin0.read_digital()   # Renvoie le niveau logique sur P0
print(val)                  # Affichage du niveau logique
```

La micro :bit intègre deux boutons notés A et B respectivement avec les attributs `button_a` et `button_b` . Ils sont connectés à P5 et P11.

```
from microbit import *
val = button_a.is_pressed() # renvoie True ou False
print(val)                  # Affichage de l'état du bouton A
```

### 4.2.5 Générer une tension MLI (PWM)

Comme avec Arduino, il est possible de générer une tension Modulée en Largeur d'Impulsion (MLI ou PWM en anglais) avec la fonction `write_analog(duty)`. Le paramètre `duty` est le rapport cyclique codé sur 12 bits (de 0 à 1023 pour un rapport cyclique de 0 à 100%).

La fréquence du signal est fixée par les fonctions `set_analog_period(T)` ou `set_analog_period_microseconds(T)` où `T` est la période respectivement en millisecondes et microsecondes.

```
from microbit import *
Pin0.set_analog_period(100)  # fixe une période de 100 ms
pin0.write_analog(767)       # rapport cyclique à 75% sur P0
```

**Note**

Il est intéressant ici de mesurer la tension moyenne au voltmètre numérique (entre GND et P0) en position DC.

### 4.2.6 Mesurer une tension (CAN)

La lecture sur 12 bits d'une tension entre 0 V et 3,3 V est effectuée par la méthode `read_analog()`.

```
from microbit import *
val = Pin0.read_analog() # renvoie un nombre entre 0 à 1023
print(val*3.3/1023)      # affichage de la tension
```

### 4.2.7 Générer une tension analogique (CNA)

La carte ne dispose pas de vraies sorties analogiques (pas de CNA) !

### 4.2.8 Faire une pause

Les fonctions `sleep(T)`, `sleep_ms(T)` et `sleep_us(T)` du module `utime` permettent de faire une pause de durée `T` respectivement en seconde, milliseconde et microseconde.

```
from utime import sleep
while True:
   Pin0.write_digital(1)
   sleep(1)
   Pin0.write_digital(0)
   sleep(1)
```

### 4.2.9 Mesurer une durée

Il est possible de mesurer la durée d'une impulsion à l'état haut ou l'état bas avec la fonction `time_pulse_us()` du module `machine` commun à tous les microcontrôleurs sous MicroPython.

**exemple**

mesurer la durée à l'état haut d'une impulsion sur l'entrée X1.

```
from microbit import *
from machine import time_pulse_us
duree = time_pulse_us(Pin0,1)
print(duree)
```

## 4.3 Langage Python avec Nanpy

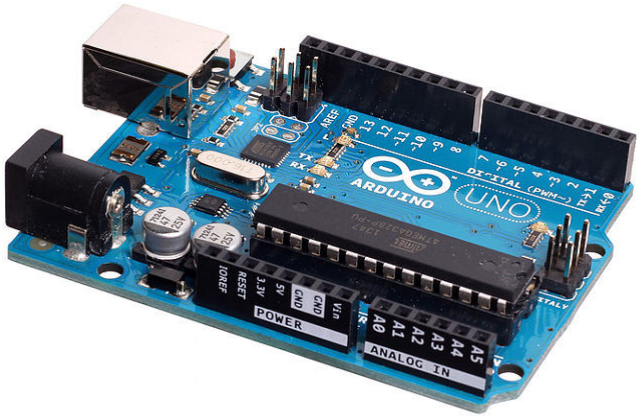



FIG. 4 – Arduino Uno R3

### 4.3.1 Qu'est-ce que Nanpy ?

Nanpy est une librairie pour Python utilisée pour le **pilotage** d'une carte Arduino **par le port USB d'un ordinateur programmé en Python**.

**Avertissement**

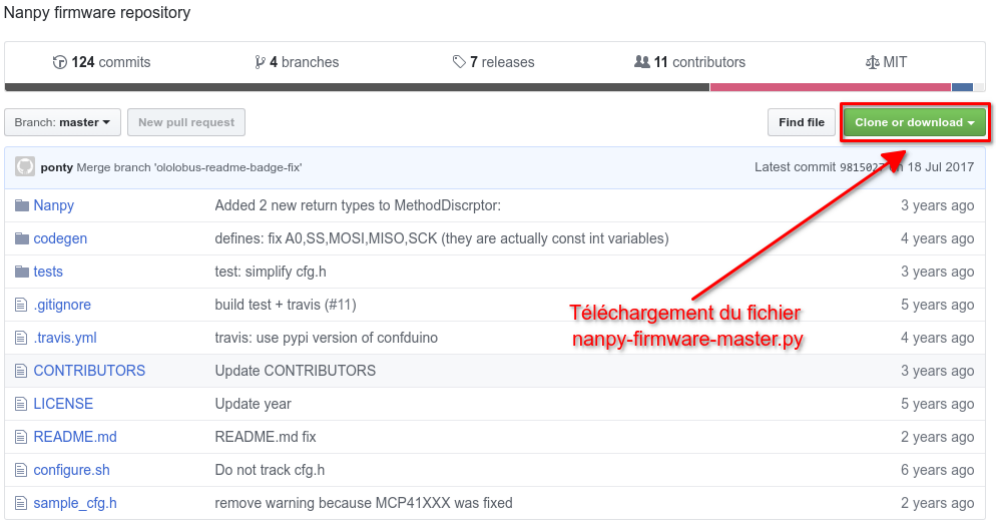
Il est important de retenir que **Nanpy ne permet pas un fonctionnement autonome** de la carte Arduino puisque le carte doit-être **constamment connectée à l'ordinateur** sur lequel le programme Python est exécuté !

### 4.3.2 Principe de fonctionnement

La carte Arduino a été préalablement programmer avec le micro-logiciel **Nanpy-firmware** téléversé à partir du logiciel Arduino. Ce firmware est un **programme particulier** (écrit en langage Arduino C/C++) qui **gère le protocole de communication** entre le programme Python exécuté sur l'ordinateur et la carte Arduino.

### 4.3.3 Téléversement du firmware Nanpy sur la carte Arduino

Il faut d'abord installer le firmware dans les croquis du logiciel Arduino. Le fichier `nanpy-firmware-master.zip` est à télécharger sur le site https://github.com/nanpy/nanpy-firmware.

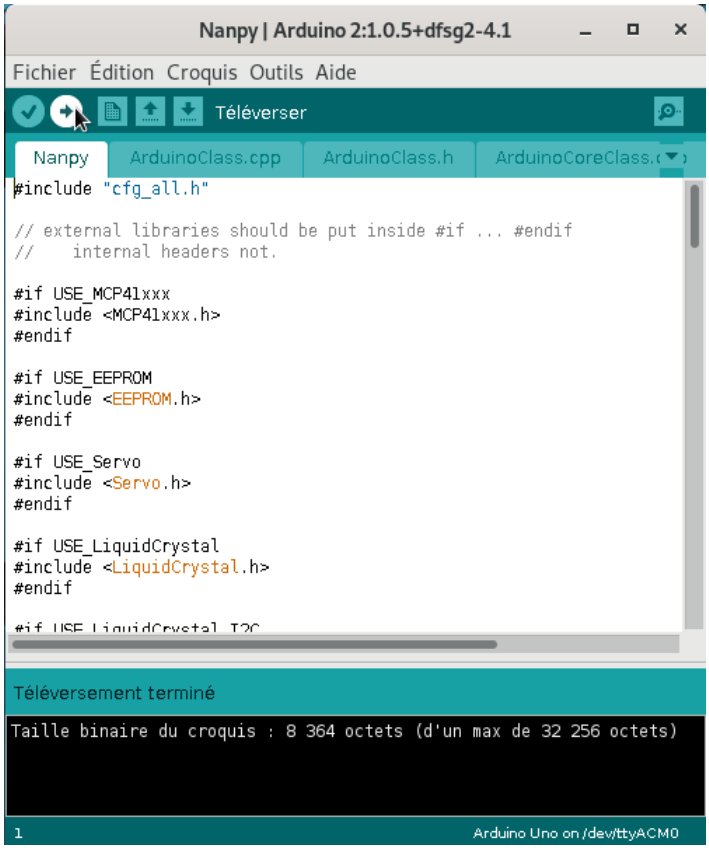



Puis il faut procéder dans l'orde suivant :

— extraire l'archive `nanpy-firmware-master.zip` dans le répertoire de travail Arduino ;
— aller dans le répertoire `nanpy-firmware-master`
— copier le fichier `sample_cfg.h` dans le répertoire `Nanpy` ;
— pour finir, renommer le nouveau fichier `sample_cfg.h` en `cfg.f`.

Ensuite, il faut procéder au téléversement du croquis du firmware sur la carte Arduino. Dans le logiciel Arduino, ouvrir le carnet de croquis `Nanpy.ino` à partir du répertoire `Nanpy`.

Puis **téléverser** le programme sur la carte Arduino.




La carte est maintenant prête pour un fonctionnement avec Nanpy !

**Avertissement**

Ne pas oublier de sélectionner le type de carte (ex. Arduino Uno) et le port de communication série (ex. COM3) dans le menu *Outils* avant le téléversement du firmware !

### 4.3.4 Installation la librairie Nanpy sur l'ordinateur

L'installation dépend de la distribution Python utilisée sur l'ordinateur. La librairie Nanpy peut-être installée par défaut.

Si ce n'est pas le cas, il faut installer manuellement à partir du dépôt internet `Pypi` (https://pypi.org/project/nanpy/) avec la commande `pip` :

```
python -m pip install nanpy
```

Ou bien à partir de l'archive Zip `nanpy-master.zip` téléchargée sur le site Github de Nanpy (https://github.com/nanpy) :

```
python -m pip install nanpy-master.zip
```

**Avertissement**

L'archive Zip doit être enregistré dans le répertoire où la commande `pip` a été exécutée.

### 4.3.5 Exemple : le programme Blink

Voici un exemple du programme **Blink** en Python.

```
from nanpy import ArduinoApi, SerialManager   # Fonctions de Nanpy
from time import sleep                        # Importation fonction sleep()

port = SerialManager(device='COM3')   # Sélection du port série (exemple : device =
 ↪'COM6')
uno = ArduinoApi(connection=port)     # Déclaration de la carte Arduino Uno

pinLed = 13                           # Led intégrée sur broche 13
uno.pinMode(pinLed,uno.OUTPUT)        # Broche Led en sortie

for i in range(100):                  # Boucle : répéter 100 fois
   uno.digitalWrite(pinLed,1)         # Led allumée
   sleep(1)                           # Attendre 1 s
   uno.digitalWrite(pinLed,0)         # Led eteinte
   sleep(1)                           # Attendre 1 s
```

# Chapitre 5

# Les bases d'Arduino

## 5.1 Allumer une LED (sorties numériques)

### 5.1.1 Principe

Les entrées/sorties numériques de la carte Arduino UNO sont accessibles sur les broches 0 à 13.

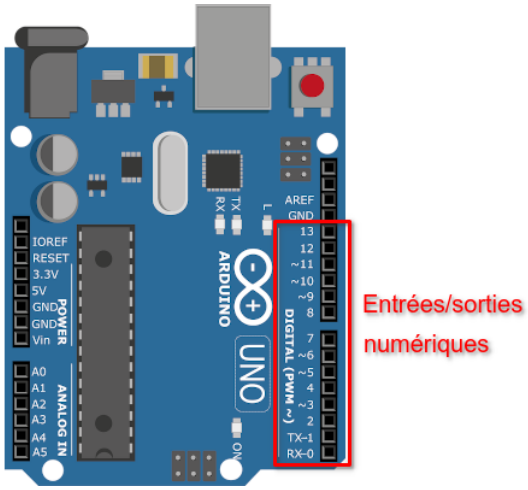



FIG. 1 – Entrées/sorties numériques

> **Avertissement**
>
> Une sortie numérique ne peut délivrer que 40 mA au maximum (200 mA pour l'ensemble des sorties). Au delà de cette valeur, la carte peut-être détériorée !

### 5.1.2 Montage

Une LED en série avec une résistance est connectée entre la broche 11 et la masse (GND).

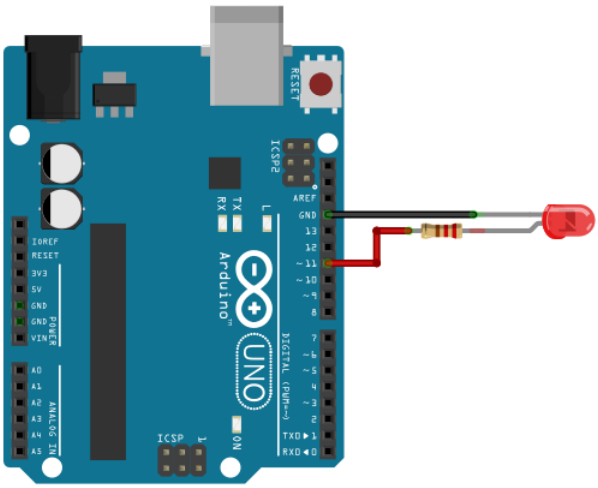


FIG. 2 – Branchement d'une LED sur la broche 11

La formule suivante donne le calcul de la résistance en fonction du courant nominal et de la couleur de la LED :

$$R = \frac{V_{CC} - V_S}{I}$$

— $V_{CC}$ est la tension d'alimentation (5 V) ;
— $V_S \approx 2V$ est la tension de seuil de la LED (dépend de la couleur) ;
— $I$ est l'intensité du courant généralement de l'ordre de 20 mA.

Une valeur de résistance de 220 Ω donne un bon compromis.

### 5.1.3 Programme en langage Arduino (C/C++)

```
int pinLED = 11;              // LED connectée sur broche 11

void setup() {
  pinMode(pinLED, OUTPUT);    // Broche LED paramétrée en sortie
}

void loop() {
  digitalWrite(pinLED,0);     // LED éteinte
  delay(1000);                // Attendre 1000 ms = 1s
  digitalWrite(pinLED,1);     // LED allumée
  delay(1000);                // Attendre 1000 ms = 1s
}
```

Dans la fonction `setup()` :

— `pinMode(LED,OUTPUT)` paramètre la broche `LED` en sortie (`OUPUT`).

Dans la fonction `loop()` :

— `digitalWrite(LED,LOW)` fixe un niveau logique 0 (`LOW`) sur la broche `LED`.
— `digitalWrite(LED,HIGH)` fixe un niveau logique 1 (`HIGH`) sur la broche `LED`.

### 5.1.4 Programme en langage Python (Nanpy)

```
from nanpy import ArduinoApi, SerialManager
from time import sleep                          # Importation fonction sleep()

port = SerialManager(device='COM6')             # Sélection du port série à modifier
uno = ArduinoApi(connection=port)               # Déclaration de la carte Arduino Uno

pinLed = 11                                     # Led branchée sur broche 11
uno.pinMode(pinLed, uno.OUTPUT)                 # Broche Led en sortie

for i in range(100):                        # Boucle : répéter 100 fois
   uno.digitalWrite(pinLed, 0)              # Led éteinte
   sleep(1)                                 # Attendre 1 s
   uno.digitalWrite(pinLed, 1)              # Led allumée
   sleep(1)                                 # Attendre 1 s
```

— La fonction `SerialManager()` fixe le port série sur lequel l'Arduino est connecté.
— La fonction `ArduinoApi()` déclare un objet (ici `uno`) qui représente la carte Arduino.
— La méthode `pinMode()` fixe la broche *pinLed* en sortie (`uno.OUTPUT`).
— La méthode `digitalWrite()` écrit un niveau logique sur la broche `pinLed` (0 pour 0 V et 1 pour 5 V).

### 5.1.5 Applications

— Commande d'un actionneur (LED, relais, ...) en tout ou rien.
— Communication numérique.

## 5.2 Modifier l'intensité lumineuse d'une LED (sorties PWM)

### 5.2.1 Principe

La carte Arduino ne possède pas de vraies sorties analogiques. Mais il est possible de faire **varier la valeur moyenne de la tension d'une sortie digitale** (donc de faire varier l'intensité lumineuse d'une LED) en modifiant son **rapport cyclique** (durée de l'état haut par rapport à la période). C'est le principe de la Modulation à Largeur d'Impulsion (MLI) ou Pulse Width Modulation (PWM) en anglais.

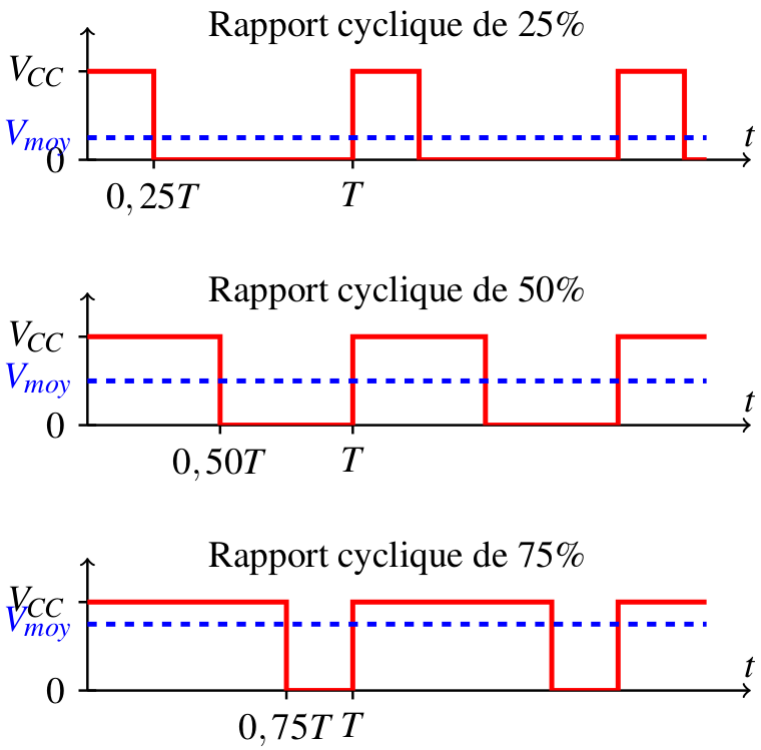



FIG. 3 – Principe de la MLI ou PWM

La carte Arduino UNO dispose de 6 sorties PWM sur les broches 3, 5, 6, 9, 10, 11.

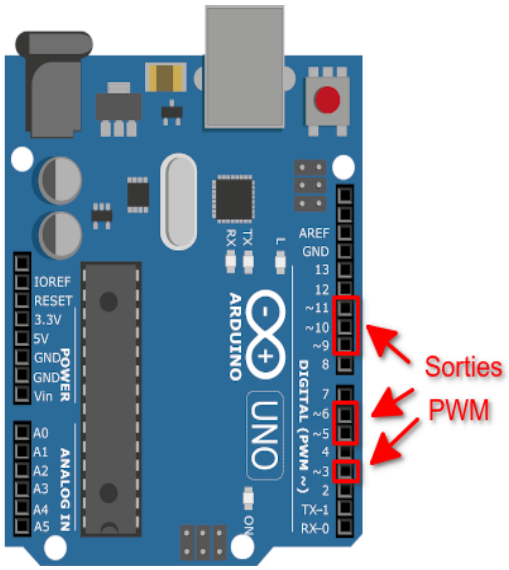



FIG. 4 – Soties PWM de l'Arduino Uno R3

**Avertissement**

La fréquence d'un signal PWM est fixée à 490 Hz pour un arduino Uno R3 !

### 5.2.2 Montage

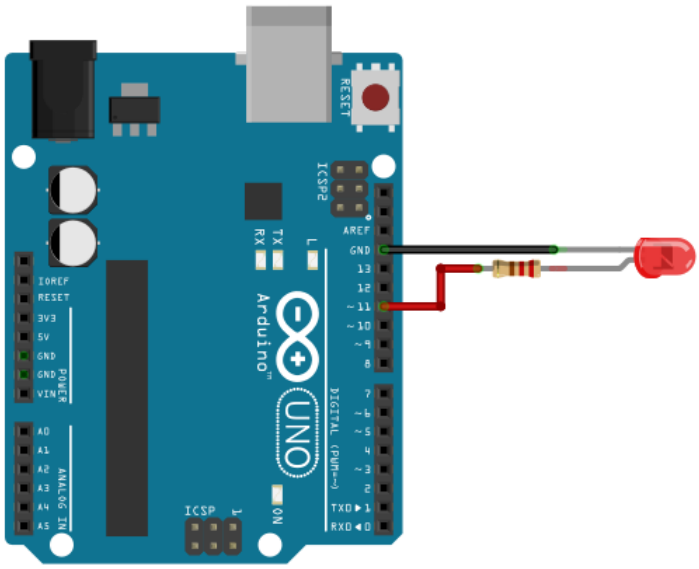


FIG. 5 – Branchement d'une LED sur la broche 11

Une LED en série avec une résistance de 220 Ω est branchée sur la broche 11.

### 5.2.3 Programme en langage Arduino (C/C++)

```
#define LED 11            // LED connectée à la broche 11

void setup() {
  pinMode(LED,OUTPUT);    // Configuration de la broche LED en sortie
}

void loop() {
  analogWrite(LED,0);     // PWM à 0%
  delay(1000);            // Attendre 1 s
  analogWrite(LED,100);   // PWM à 100/255 = 39%
  delay(1000);            // Attendre 1 s
  analogWrite(LED,200);   // PWM à 200/255 = 78%
  delay(1000);            // Attendre 1 s
}
```

La fonction `analogWrite(LED,duty)` génère une modulation à largeur d'impulsion sur la broche 11 où `duty` est un nombre entier entre 0 et 255 respectivement pour un rapport cyclique entre 0% et 100%.

Le code qui suit est une version avec la saisie du rapport cyclique au clavier dans le moniteur série.

```
// PWM avec saisie du rapport cyclique (entier de 0 à 255) au clavier dans le moniteur␣
 ↪série.
// ATTENTION : Sélectionner "Pas de fin de ligne" dans le monitor série !!!
// David THERINCOURT 2025

#define LED 11            // LED connectée à la broche 11

void setup()
```

(suite sur la page suivante)

(suite de la page précédente)

```
{
   Serial.begin(9600);   // Initialisation du port série
   pinMode(LED,OUTPUT);  // Configuration de la broche LED en sortie
}

void loop()
{
   Serial.print("Rapport cyclique entre 0 et 255 : ");  // Indication à l'utilisateur
   while (Serial.available()==0){}                      // Attente d'un message (Cocher
 ↪"Pas de fin de ligne")
   int N = Serial.parseInt();                           // Extraction de la valeur␣
 ↪numérique (entier)
   Serial.println(N);                                   // Affichage de N
   analogWrite(LED, N);                                 // Ecriture sur la sortie PWM
}
```

### 5.2.4 Programme en langage Python (Nanpy)

```
from nanpy import ArduinoApi, SerialManager
from time import sleep

port = SerialManager(device='COM6')          # Sélection du port série à modifier
uno = ArduinoApi(connection=port)            # Déclaration de la carte Arduino Uno

pinLed = 11                                  # Led branchée sur broche 11
uno.pinMode(pinLed, uno.OUTPUT)              # Broche Led en sortie

for i in range(9):
    uno.analogWrite(pinLed, 10)   # PWM à 10/255
    sleep(1)                      # Attendre 1s
    uno.analogWrite(pinLed, 50)   # PWM à 50/255
    sleep(1)                      # Attendre 1s
    uno.analogWrite(pinLed, 200)  # PWM à 200/255
    sleep(1)                      # Attendre 1s

port.close()                      # Fermeture du port série
```

#### 5.2.4.1 Applications

— Variation de l'intensité lumineuse d'une LED.
— Variation de la vitesse d'un moteur à courant continu.
— Obtention d'une tension constante par filtrage passe-bas (limitée en fréquence).

## 5.3 Afficher des messages (port série UART)

### 5.3.1 Principe

De base, les cartes Arduino ne possède pas d'écran pour afficher des messages. L'interface série (UART) reste le moyen le plus simple pour afficher des informations sur un ordinateur en provenance d'une carte Arduino.

### 5.3.2 Programmation en langage Arduino (C/C++)

Le logiciel Arduino IDE met à disposition un **moniteur série** (dans le menu `Outils > Moniteur série`) pour **lire des données** au format texte (ASCII) envoyées par le microcontrôleur.

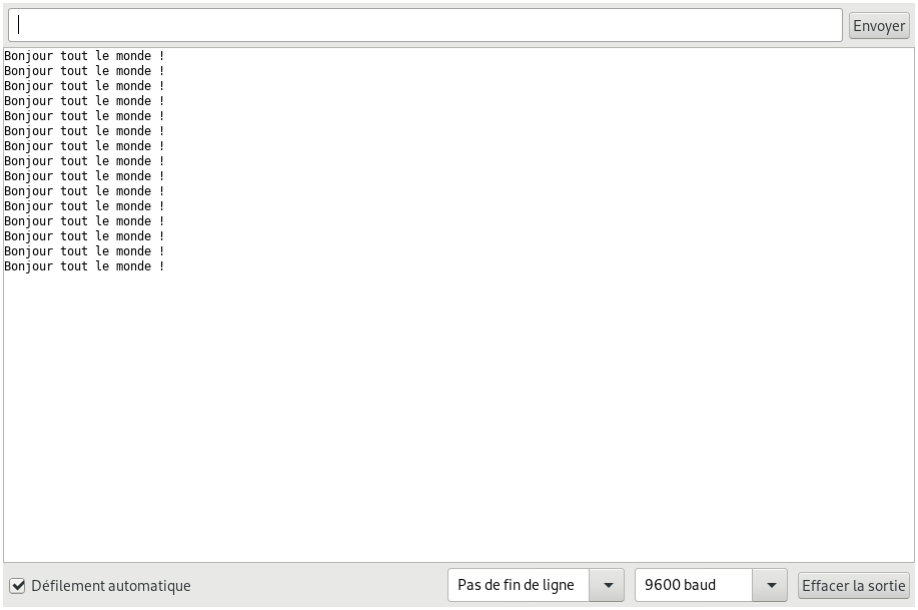



FIG. 6 – Moniteur série du d'Arduino IDE 1.8

Le moniteur série permet également de transmettre des données vers le microcontrôleur !

**Note**

Il est possible d'utiliser d'autres logiciels de communication série comme Putty ou encore Termite.

```
int n = 0;

void setup() {
  Serial.begin(9600);     // Paramétrage du port série

}

void loop() {
  Serial.print("N = ");    // Affichage
  Serial.println(n);       // Affichage du contenu de la variable n
  n = n + 1;               // Incrémentation de la variable n
  delay(1000);             // Temporisation de 1s
}
```

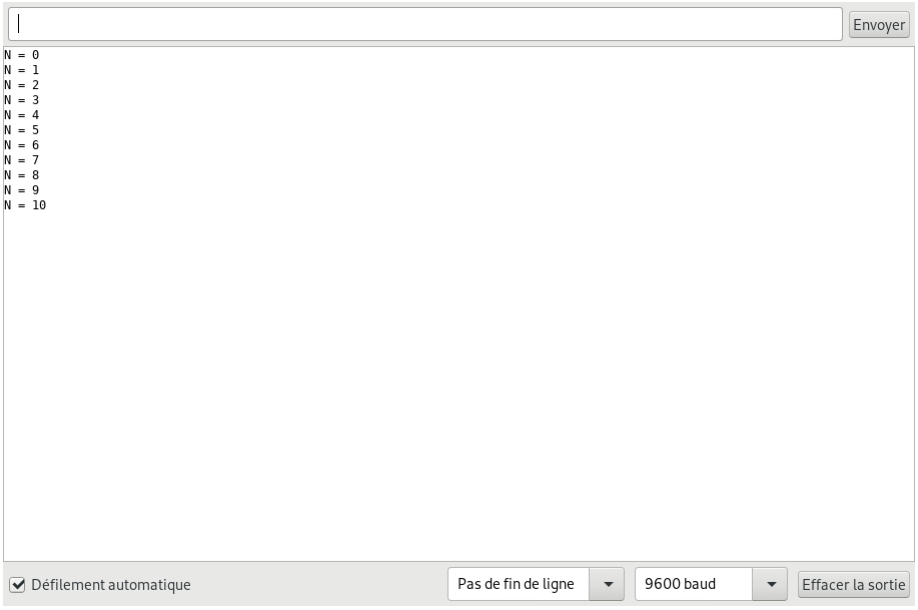



FIG. 7 – Affichage dans le moniteur série

— L'instruction `Serial.begin(9600)` paramètre le port série à 9600 baud (bits par seconde).
— `Serial.print("N = ")` affiche la chaine de caractères `N =` dans le moniteur série.
— `Serial.println(n)` affiche le contenu de la variable `n` suivie d'un saut de ligne.

### 5.3.3 Programmation en langage Python (pilotage Nanpy)

Avec Nanpy, le code Python est exécuté sur l'ordinateur et non sur le microcontrôleur. La fonction native `print()` de Python s'utilise tout naturellement !

```
from nanpy import ArduinoApi, SerialManager
from time import sleep

port = SerialManager(device='COM6')  # Sélection du port série à modifier
uno = ArduinoApi(connection=port)    # Déclaration de la carte Arduino Uno

N = 0                                # Initialisation du compteur

while True:
   print("N =", N)                   # Affichage
   N = N + 1                         # Incrémentation du compteur
   sleep(1)                          # Temporisation
```

### 5.3.4 Applications

— Affichage d'une ou plusieurs mesures sur l'écran d'un ordinateur.
— Affichage des données d'une acquisition au format CSV pour exploitation par un tableur ou des logiciels spécialisés (Regressi, Latis, ...).

## 5.4 Mesurer une tension (CAN)

### 5.4.1 Principe

La mesure d'une tension par un microcontrôleur est réalisée en interne par un **convertisseur analogique-numérique**.

Ce type de conversion est importante en sciences physique. Elle permet par exemple d'obtenir la mesure d'une grandeur physique provenant d'un **capteur**.

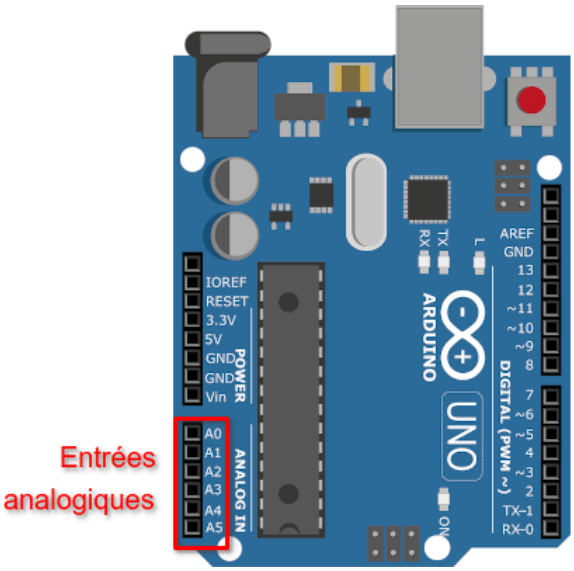



FIG. 8 – Entrées analogiques de l'Arduino Uno R3

Arduino UNO dispose de **6 entrées analogiques** disponibles sur les broches A0 à A5. Par défaut, la tension en entrée doit-être comprise entre 0 V et 5 V (Vref). Le résultat de la conversion donne un nombre sur 10 bits compris entre 0 et 1023. La résolution analogique (**quantum**) est donc :

$$q = \frac{5}{1023} \approx 4,89\,mV$$

**Avertissement**

Le tension appliquée sur les entrées analogiques doivent être **strictement comprise entre 0 V et 5 V** sous peine de détruire le microcontrôleur.

### 5.4.2 Montage

Un potentiomètre monté en pont diviseur de tension est branché entre la masse (`GND`) et la tension d'alimentation (`5V`). Ce potentiomètre délivre donc une **tension réglable** en `0 V` et `5 V` sur l'entrée `A0`.

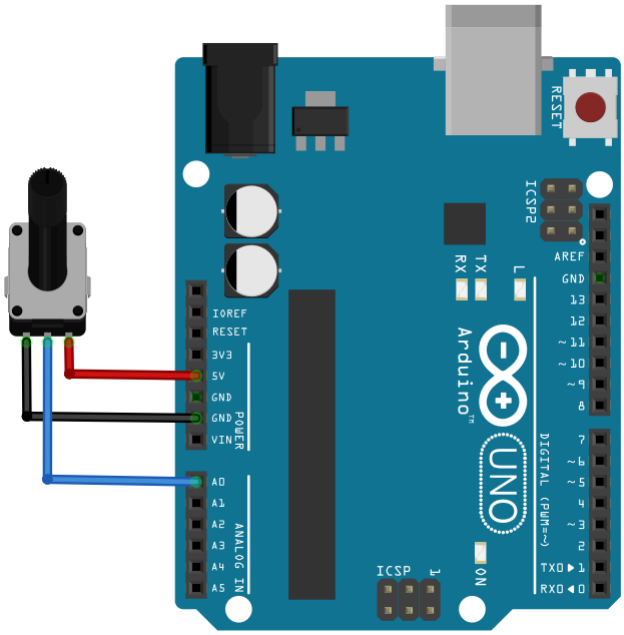



FIG. 9 – Montage potentiométrique sur l'entrée analogique A0

### 5.4.3 Programme en langage Arduino (C/C++)

Le programme suivant lit la tension sur l'entrée `A0` et affiche sa valeur dans le moniteur série du logiciel Arduino.

```
/*
 * Lecture de l'entrée analogique A0 sur le port série.
 */

int N;           // Entier lu sur A0 compris entre 0 et 1023 (10bits)
float tension;   // La tension calculée à partir de N

void setup() {
  Serial.begin(9600);   // Paramétrage port série
}

void loop() {
  N = analogRead(A0);                    // Lecture valeur sur A0
  tension = N*5.0/1023;                  // Calcul de la tension
  Serial.print("Valeur lue sur A0 = ");
  Serial.println(N);                     // Affichage valeur sur A0
  Serial.print("Tension = ");
  Serial.println(tension);               // Affichage de la tension
  delay(1000);                           // temporisation de 1 s
}
```

— La fonction `analogRead(A0)` retourne un entier sur 10 bits compris entre 0 (pour 0V) et 1023 (pour 5V).
— L'expression `N*5.0/1023` calcule la valeur de la tension en volt.

### 5.4.4 Programme en langage Python (pilotage Nanpy)

```
from nanpy import ArduinoApi, SerialManager
from time import sleep                          # Importation fonction sleep()

port = SerialManager(device='COM6')             # Sélection du port série à modifier
uno = ArduinoApi(connection=port)               # Déclaration de la carte Arduino Uno


for i in range(10):
   N = uno.analogRead(0)                  # Lecture la tension numérique sur A0
   print("N = ", N)                       # Affichage
   U = N*5/1023                           # Calcul de la tension en volt
   print("U = ", round(U, 3), " V")       # Affichage
   sleep(1)                               # Temporisation d'une seconde

uno.connection.close()                     # Deconnexion de Arduino
port.close()                               # Fermeture du port série
```

### 5.4.5 Applications

— Interface avec un circuit comportant un capteur.
— Un potentiomètre est un capteur de position.

### 5.4.6 Aller plus loin : contrôler l'intensité lumineuse d'une LED

En combinant le programme précédant avec celui sur la génération d'une tension PWM, il est possible de régler à l'aide d'un potentiomètre l'intensité lumineuse d'une LED.

#### 5.4.6.1 Montage

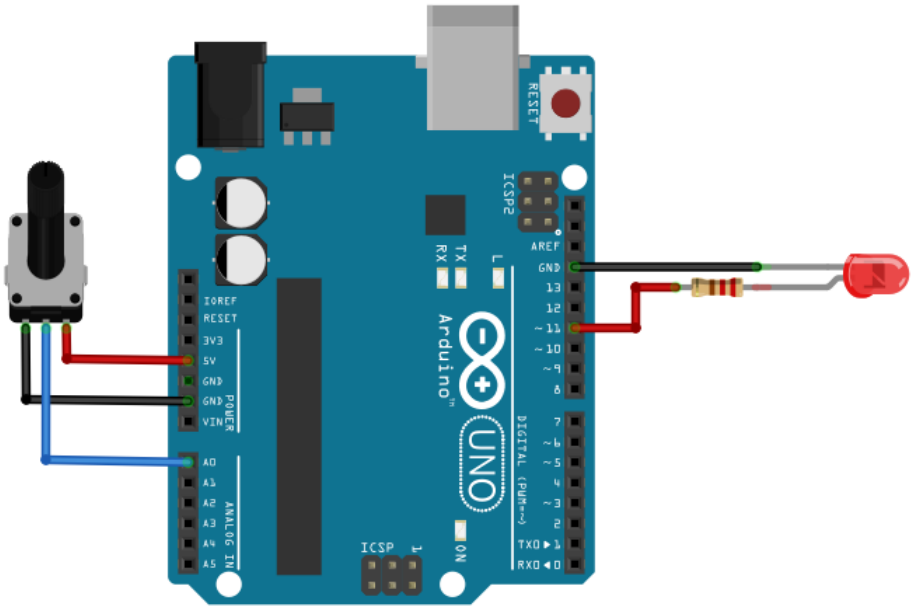



FIG. 10 – Commande de l'intensité d'une LED avec un potentiomètre

#### 5.4.6.2 Programme en langage Arduino (C/C++)

```
#define pinLED 11

int N;       // Valeur lue sur A0 de 0 à 1023
int duty;    // Rapport cyclique de 0 à 255

void setup() {
}

void loop() {
  N = analogRead(A0);       // Conversion analogique-numérique sur A0
  duty = N/4;               // Calcul du rapport cyclique
  analogWrite(pinLED, duty); // Génération de la tension PWM
  delay(30);                // Attendre 30 ms
}
```

**Note**

Pour convertir un entier sur 10 bits en un entier sur 8 bits, il suffit de la **division entière par 4** !

#### 5.4.6.3 Programme en langage Python (pilotage Nanpy)

```
from nanpy import ArduinoApi, SerialManager

port = SerialManager(device='COM6')          # Sélection du port série à modifier
uno = ArduinoApi(connection=port)            # Déclaration de la carte Arduino Uno

pinLed = 11

while True:
    N = uno.analogRead(0)               # Lecture la tension numérique sur A0
    duty = N//4                         # Division entière par 4
    uno.analogWrite(pinLed, duty)       # Tension PWM sur la LED
```

# Chapitre 6

# Nouveaux programmes du lycée

## 6.1 Capteur résistif - CTN (seconde générale)

**Programme de seconde générale 2019 - Enseignement commun**

Mesurer une grandeur physique à l'aide d'un capteur électrique résistif. **Produire et utiliser une courbe d'étalonnage** reliant la résistance d'un système avec une grandeur d'intérêt (température, pression, intensité lumineuse, etc.). Utiliser un dispositif avec microcontrôleur et capteur.

### 6.1.1 Cas d'une CTN

Une CTN est un **capteur résistif à coefficient de température négatif** dont l'évolution de la résistance en fonction de la température est donnée par la figure suivante :

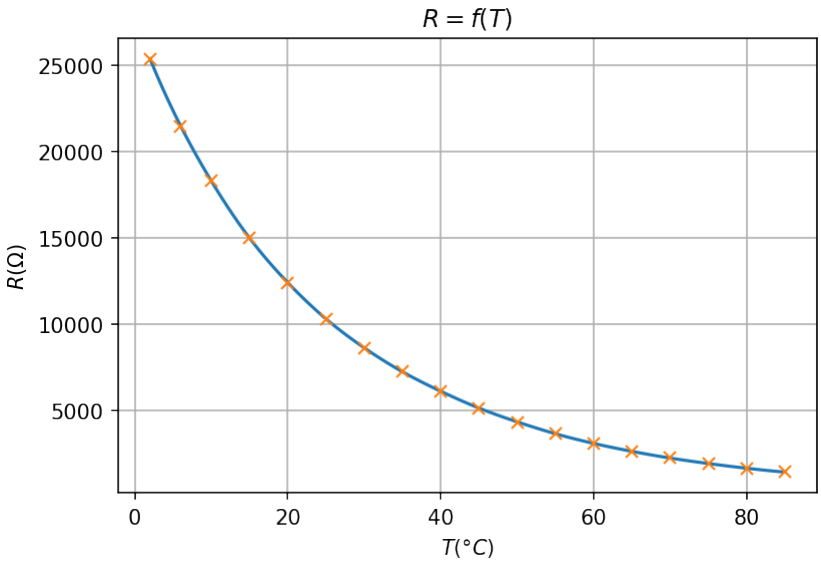
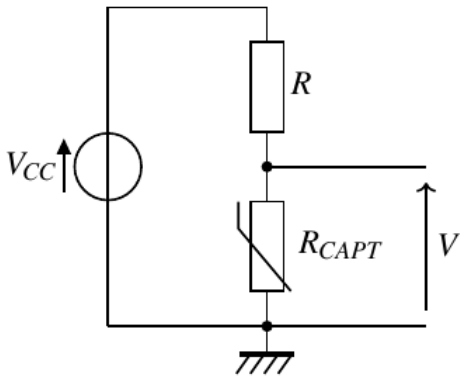



FIG. 1 – Caractéristique R=f(T) d'une CTN

### 6.1.2 Principe de mesure de résistance de la CTN

La plupart des modules pour capteur résistif utilise un **pont diviseur de tension** pour la mesure de la résistance du capteur. Par rapport au pont Wheatstone, cette méthode présente l'avantage d'être simple à mettre en oeuvre.

#### 6.1.2.1 Montage 1 : capteur connecté à la masse




$$R_{CAPT} = R \times \frac{V}{V_{CC} - V}$$

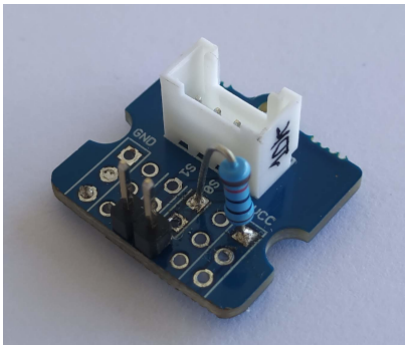
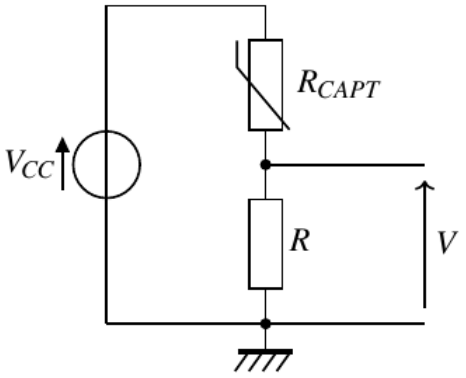



FIG. 2 – Module personnel Grove ($R = 10$ kΩ)

#### 6.1.2.2 Montage 2 : capteur connecté à Vcc




$$R_{CAPT} = R \times (\frac{V_{CC}}{V} - 1)$$

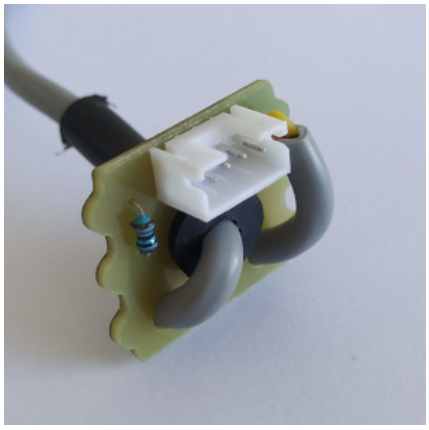
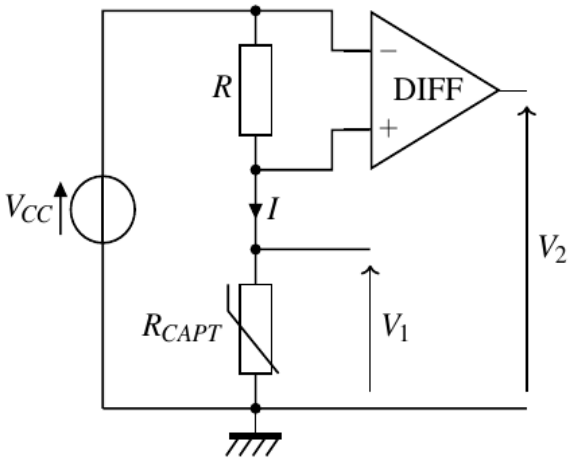


FIG. 3 – Module CTN Plug'uino ($R = 10$ kΩ)

#### 6.1.2.3 Montage : mesure de la tension et du courant (ex. capteurs Educaduino)




$$I = \frac{V_2}{R}$$

$$R_{CAPT} = \frac{V_1}{I}$$

$$R_{CAPT} = R \times \frac{V_1}{V_2}$$

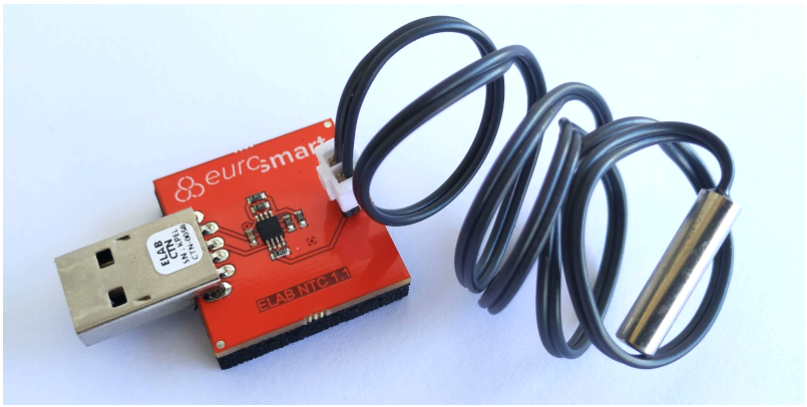



FIG. 4 – Module CTN Educaduino LAB ($R = 10$ kΩ)

En plus de la **mesure de la tension** du capteur, une **mesure du courant** est aussi réalisée à partir de la tension aux bornes de la résistance R par l'intermédiaire d'un amplificateur différentiel. La résistance du capteur est alors calculée avec la **loi d'Ohm**.

### 6.1.3 Mesure de la résistance de la CTN

L'ensemble formé du module capteur résistif et du microcontrôleur est vu comme un **ohmmètre**. Les programmes suivants ont donc pour objectif de mesurer la résistance de la CTN.

#### 6.1.3.1 Arduino (C/C++)

L'entrée analogique `A0` mesure la tension du capteur.

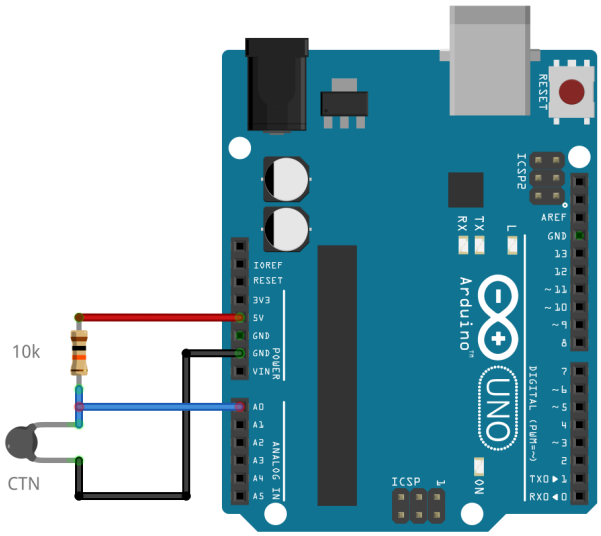



FIG. 5 – Branchement d'une CTN sur l'entrée analogique A0

```
// Mesure de la résistance d'une CTN

#define Vcc 5       // Tension d'alimentation
#define Ro  10000   // Résistance du pont

float U;            // Tension CTN
float R;            // Résistance CTN


void setup() {
   Serial.begin(9600);  // Paramétrage du port série
}

void loop() {
  U = analogRead(A0)*5.0/1023;      // Lecture tension en V
  R = Ro*U/(Vcc-U);                 // Calcul de la résistance
  Serial.println(R);                // Affichage
  delay(1000);                      // Temporisation de 1s
}
```

#### 6.1.3.2 Arduino (Python/Nanpy)

Le montage reste le même.

```
# Mesure de la résistance d'une CTN
from nanpy import ArduinoApi            # Gestion de l'Arduino
from nanpy import SerialManager         # Gestion port série
from time import sleep                  # Importation de sleep(seconde)

Vcc = 5.0                               # Tension d'alimentation
Ro = 10000                              # Résistance du pont

port = SerialManager(device='COM6')     # Sélection du port série (à remplacer)
uno = ArduinoApi(connection=port)       # Déclaration de la carte Arduino


while True :
   U = uno.analogRead(0)*5/1023         # Lecture la tension sur A0
   R = Ro*U/(Vcc-U)                     # Calcul de la résistance
   print("R = ", R)                     # Affichage
   sleep(1)                             # Temporisation d'une seconde

port.close()                            # Fermeture du port série
```

#### 6.1.3.3 PyBoard (MicroPython)

Le montage ci-dessous utilise une carte Feather STM32F405 Express. L'entrée analogique A0 mesure la tension du capteur.

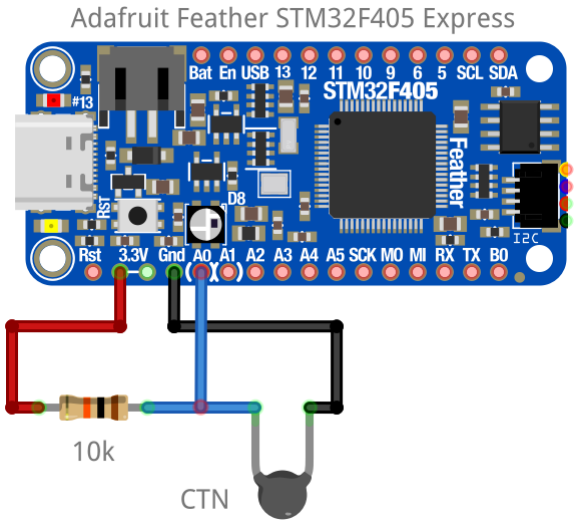



FIG. 6 – Branchement d'une CTN sur l'entrée analogique A0

```
# Mesure de la resistance d'une CTN
from pyb import Pin, ADC, delay

adc = ADC(Pin("A0")) # Déclaration du CAN
Ro = 10e3 # Résistance série

while True:
   N = adc.read() # Mesure de la tension
   R = Ro*N/(4095-N) # Calcul de R
```

(suite sur la page suivante)

(suite de la page précédente)

```
   print("R =", R) # Affichage
   delay(1000) # Temporisation
```

#### 6.1.3.4 Micro :bit (MicroPython)

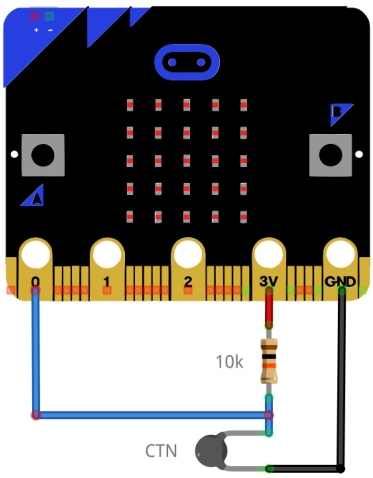



FIG. 7 – Branchement d'une CTN sur l'entrée analogique pin0

```
# Mesure de la resistance d'une CTN
from microbit import *

Ro = 10e3                    # Résistance série

while True:
   N = pin0.read_analog()    # Mesure de la tension
   R = Ro*N/(1023-N)         # Calcul de R
   print("R =", R)           # Affichage
   sleep(1000)               # Temporisation
```

### 6.1.4 Caractéristique R=f(T) de la CTN

#### 6.1.4.1 Courbe d'étalonnage

Les mesures suivantes peuvent être effectuées avec le **microcontrôleur** ou à **l'ohmmètre**.

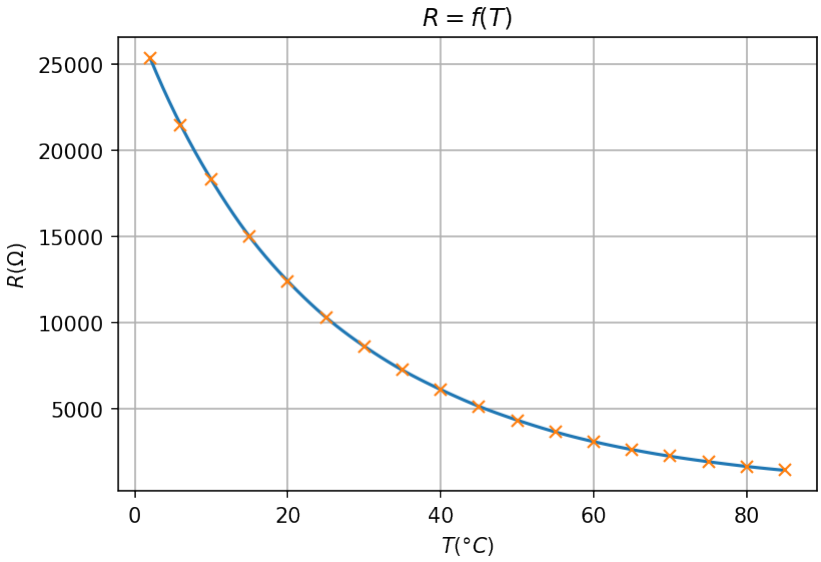



FIG. 8 – Courbe d'étalonnage d'une CTN 10k

**Note**

Dans cet exemple, la résistance mesurée prend la valeur particulière de 10 $k\Omega$ pour 25°C !

#### 6.1.4.2 Relation de Steinhart-Hart

Sur une grande plage de variation, la relation entre la température (en K) et la résistance de la CTN est :

$$\frac{1}{T} = A + B \times \ln(R) + C \times (\ln(R))^3$$

A, B et C sont les coefficients de Steinhart-Hart. Ils sont donnés par le constructeur ou peuvent se déterminer expérimentalement à l'aide du programme Python `steinhart-hart.py` à partir de trois points de la courbe d'étalonnage.

Résultats obtenus à partir du programme Python :

$$A = 1,144 \cdot 10^{-3} K^{-1} \qquad B = 2,078 \cdot 10^{-3} K^{-1} \qquad C = 3,610 \cdot 10^{-7} K^{-1}$$

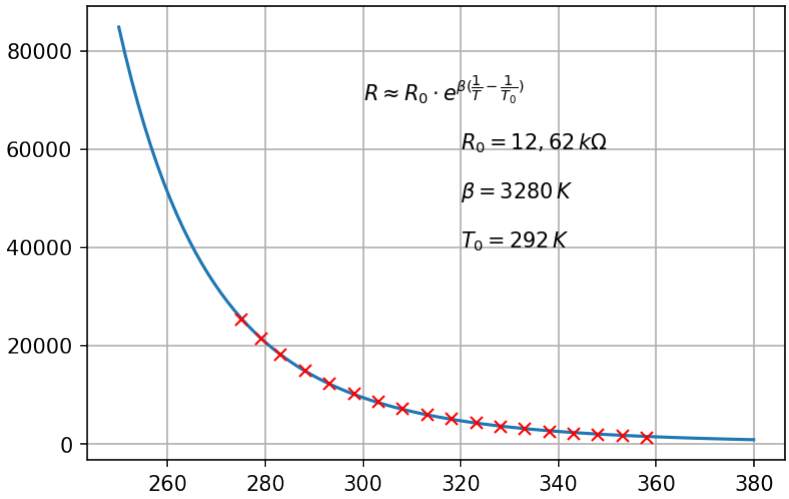
#### 6.1.4.3 Simplification de relation de Steinhart-Hart

Sur une plage de variation plus réduite de la température, la relation de Steinhart-Hart permet d'écrire :

$$R \approx R_0 \times e^{\beta(\frac{1}{T} - \frac{1}{T_0})}$$

— $R_0$ est la valeur de la résistance pour la température $T_0$.
— $\beta$ (en K).

Ces coefficients sont généralement donnés par le constructeur ou peuvent être déterminés par une modélisation de la caractéristique.




Le calcul de la température (en K) s'effectue à l'aide de la relation suivante :

$$\frac{1}{T} = \frac{1}{\beta} \times \ln(\frac{R}{R_0}) + \frac{1}{T_0}$$

### 6.1.5 Application : réaliser un thermomètre numérique

#### 6.1.5.1 Arduino (C/C++)

```
// Mesure de la resistance d'un CTN
// Calcul de la température à partir de la relation de Steinhart-Hart

#define Vcc 5       // Tension d'alimentation
#define Ro  10000   // Résistance du pont
#define A   1.0832e-3
#define B   2.1723e-4
#define C   3.2770e-7

float u;            // Tension CTN
float R;            // Résistance CTN
float logR;         // ln(R)
float T;            // Température en °C


void setup() {
   Serial.begin(9600);  // Paramétrage du port série
}

void loop() {
  u = analogRead(A0)*5.0/1023;                  // Lecture tension en V
  R = Ro * u/(Vcc-u);                           // Calcul de la résistance
  logR = log(R);                                // Calcul de ln(R)
  T = (1.0 / (A + B*logR + C*logR*logR*logR));  // Calcul de la température
  T = T - 273.15;                               // Conversion en °C
  Serial.print("R = ");                         // Début affichage
  Serial.println(R);
  Serial.print("T = ");
  Serial.println(T);                            // Fin affichage
```

(suite sur la page suivante)

(suite de la page précédente)

```
  delay(1000);                                    // Temporisation de 1s
}
```

#### 6.1.5.2 Arduino (Python/Nanpy)

```
# Mesure de la resistance d'une CTN et calcul de la température
# Calcul de la température à partir de la relation de Steinhart-Hart
from nanpy import ArduinoApi            # Gestion de l'Arduino
from nanpy import SerialManager         # Gestion port série
from time import sleep                  # Importation de sleep(seconde)
from math import log                    # Importation du logarithme népérien

Vcc = 5.0      # Tension d'alimentation
Ro = 10000     # Résistance du pont
A = 1.0832e-3  # Coeff. de Steinhart-Hart
B = 2.1723e-4  # ...
C = 3.2770e-7  # ...

port = SerialManager(device='COM6')          # Sélection du port série (à remplacer)
uno = ArduinoApi(connection=port)            # Déclaration de la carte Arduino


while True :
   U = uno.analogRead(0)*5/1023              # Lecture la tension sur A0
   R = Ro*U/(Vcc-U)                          # Calcul de la résistance
   T = 1.0 / (A + B*log(R) + C*log(R)**3)    # Calcul de la température en Kelvin
   T = T-273.15                              # Calcul de la température en Celsius
   print("R = ", R, "T = ", T)               # Affichage
   sleep(1)                                  # Temporisation d'une seconde

port.close()                                 # Fermeture du port série
```

#### 6.1.5.3 PyBoard (MicroPython)

```
# Mesure de la resistance d'une CTN et calcul de la température
# Calcul de la température à partir de la relation de Steinhart-Hart
from pyb import Pin, ADC, delay
from math import log

adc = ADC(Pin("A0"))         # Déclaration du CAN

Ro = 10e3                    # Résistance série
A = 0.0010832035972923174    # Coeff. de Steinhart-Hart
B = 0.00021723460553451255   # ...
C = 3.276999926128753e-07    # ...

while True:
   N = adc.read()                                # Mesure de la tension
   R = Ro*N/(4095-N)                             # Calcul de R
   T = 1/(A + B*log(R) + C*log(R)**3) - 273.15 # Relation de Steinhart-Hart
   print("R =", R, "T =", T)                     # Affichage
   delay(1000)                                   # Temporisation
```

#### 6.1.5.4 Micro :bit (MicroPython)

```
# Mesure de la resistance d'une CTN et calcul de la température
# Calcul de la température à partir de la relation de Steinhart-Hart
from microbit import *
from math import log

Ro = 10e3     # Résistance série
A = 1.0832e-3 # Coefficients de Steinhart-Hart
B = 2.1723e-4 # ...
C = 3.2770e-7 # ...

while True:
   N = pin0.read_analog()                          # Mesure de la tension
   R = Ro*N/(1023-N)                               # Calcul de R
   T = 1/(A + B*log(R) + C*log(R)**3) - 273.15     # Relation de Steinhart-Hart
   print("R =", R, "T =", T)                       # Affichage
   sleep(1000)                                     # Temporisation
```

### 6.1.6 A retenir

Placer un **capteur résistif** (température, pression, lumière, ...) dans un **pont diviseur de tension** reste une **solution simple pour mesurer sa résistance** à l'aide d'un microcontrôleur.

## 6.2 Émission d'un son (seconde générale)

**Programme de seconde générale 2019 - Enseignement commun**

Utiliser un dispositif comportant un microcontrôleur pour produire un signal sonore.

### 6.2.1 Principe

Certains microcontrôleurs (Arduino et Micro :bit) ne possédant pas de sortie analogique (CNA) pour générer des tensions sinusoïdales, une méthode simple pour produire un son est de générer une tension carrée (entre 0 et Vcc) de fréquence f à l'entrée d'un haut-parleur.

Le son obtenu par cette technique n'est pas pur car il comporte des harmoniques aux fréquences 3f, 5f, 7f, ...

**Avertissement**

Pour les faibles fréquences, le son devient « métallique » avec la présence importante d'harmoniques !

### 6.2.2 Méthode 1 : construire le signal carré

#### 6.2.2.1 Arduino (C/C++)

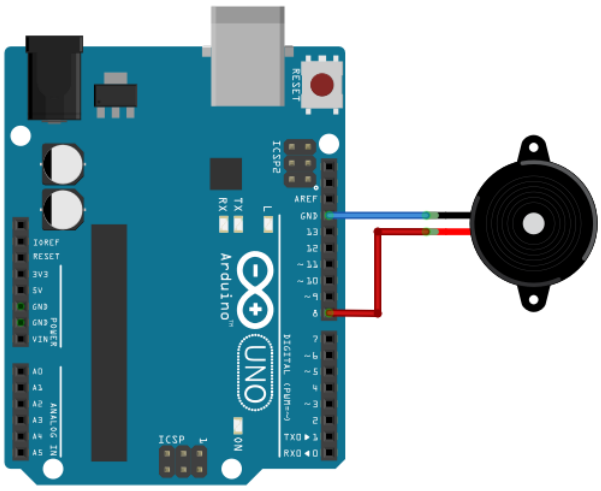


FIG. 9 – Branchement d'un haut-parleur sur la broche 8

```
#define brocheHP 8

float frequence=440;
float periode=1/frequence;

void setup(){
    pinMode(brocheHP, OUTPUT);
}

void loop(){
  digitalWrite(brocheHP,HIGH);
  delayMicroseconds(1000000*periode/2.0);
  digitalWrite(brocheHP,LOW);
  delayMicroseconds(1000000*periode/2.0);
}
```

#### 6.2.2.2 Arduino (Python/Nanpy)

Cette méthode n'est pas applicable ici car le pilotage de l'Arduino par Nanpy est **trop lent** !

### 6.2.3 Méthode 2 : utiliser une fonction spéciale

#### 6.2.3.1 Arduino (C/C++)

```
/*
Exemple Arduino
*/
#include "pitches.h"

// notes in the melody:
```

(suite sur la page suivante)

(suite de la page précédente)

```
int melody[] = {NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_C4};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {4, 8, 8, 4, 4, 4, 4, 4 };

void setup() {
for (int thisNote = 0; thisNote < 8; thisNote++) {
   int noteDuration = 1000/noteDurations[thisNote];
   tone(8, melody[thisNote],noteDuration);
   int pauseBetweenNotes = noteDuration * 1.30;
   delay(pauseBetweenNotes);
   noTone(8);
}
}

void loop() {
// Pas de boucle ici !
}
```

#### 6.2.3.2 Arduino (Python/Nanpy)

La classe `Note` de Nanpy dispose des méthodes :

— `play(fréquence, durée)` pour jouer une note pendant un durée en ms ;
— `stop()` pour arrêter la lecture de la note.

```
# Nanpy v0.96
from nanpy import ArduinoApi, SerialManager, Tone
from time import sleep

port = SerialManager(device='/dev/ttyACM0')  # Sélection du port série (exemple :␣
 ↪device = 'COM6')
uno = ArduinoApi(connection=port)            # Déclaration de la carte Arduino Uno

note =  [Tone.NOTE_C4, Tone.NOTE_G3, Tone.NOTE_G3, Tone.NOTE_A3,
         Tone.NOTE_G3, 0 , Tone.NOTE_B3, Tone.NOTE_C4]
noteDuration = [4, 8, 8, 4,
                4, 4, 4, 4]

hp = Tone(8)                                 # Haut parleur sur broche 8

for i in range(8):
   duree = 1000/noteDuration[i]              # durée en ms
   hp.play(note[i] , duree)                  # jouer la note
   sleep(1.3*duree*1E-3)                     # pause en les notes

hp.stop()                                    # Arrêt de la lecture
```

**Note**

Une méthode `tone()` équivalent à celle du langage Arduino a été ajoutée dans la version modifiée de Nanpy d'Eurosmart.

```
# Version modifiée de Nanpy par Eurosmart
from nanpy import ArduinoApi, SerialManager, Tone
from time import sleep

port = SerialManager(device='/dev/ttyACM0')  # Sélection du port série (exemple :␣
 ↪device = 'COM6')
uno = ArduinoApi(connection=port)            # Déclaration de la carte Arduino Uno

melody =  [Tone.NOTE_C4, Tone.NOTE_G3, Tone.NOTE_G3, Tone.NOTE_A3, Tone.NOTE_G3, 0 ,␣
 ↪Tone.NOTE_B3, Tone.NOTE_C4]
noteDuration = [4, 8, 8, 4, 4, 4, 4, 4]

pinHP = 8                            # Haut-parleur sur broche 8

for i in range(8):
    duree = 1/noteDuration[i]        # durée en ms
    uno.tone(pinHP, melody[i])       # Lecture de la note
    sleep(duree)                     # Attendre la lecture
    uno.noTone(pinHP)                # Arrêt de la note
    sleep(duree*1.3)                 # Pause entre les notes
```

### 6.2.4 Applications

— Hauteur d'un son (relation entre fréquences et notes).
— Générer une mélodie à partir de plusieurs notes.

## 6.3 Mesurer la célérité d'un son (première générale)

**Programme de première générale 2019 - Enseignement de spécialité.**

Exploiter la relation entre la durée de propagation, la distance parcourue par une perturbation et la célérité, notamment pour localiser une source d'onde. Déterminer, par exemple à l'aide d'un microcontrôleur ou d'un smartphone, une distance ou la célérité d'une onde.

### 6.3.1 Présentation du module HC-SR04

Les modules du type HC-SR04 sont des émetteurs-récepteurs ultrasonores fonctionnant par réflexion. Ils sont utilisés généralement dans des applications comme télémètre (< 5 m).


FIG. 10 – Module HC-SR04

#### 6.3.1.1 Fonctionnement

— Le module est alimenté entre `GND` et `Vcc` (généralement 5 V ou 3,3 V sur certains modules).
— Le déclenchement d'une mesure (émission d'une salve) se fait par une brève impulsion ($> 10$ µs) sur l'entrée `trig`.
— La durée que prend l'onde pour aller de l'émetteur au récepteur est celle de l'impulsion renvoyée sur la sortie `echo`.

#### 6.3.1.2 Modification du module

Tel qu'il est vendu, ce module n'a peu d'intérêt en sciences physiques car les signaux électriques sur l'émetteur et le récepteur ne sont pas accessibles. Il est possible de résoudre ce problème en y soudant des **connecteurs supplémentaires** (voir photo ci-dessous) et de **visualiser les signaux correspondants à l'oscilloscope ou avec une interface d'acquisition**.

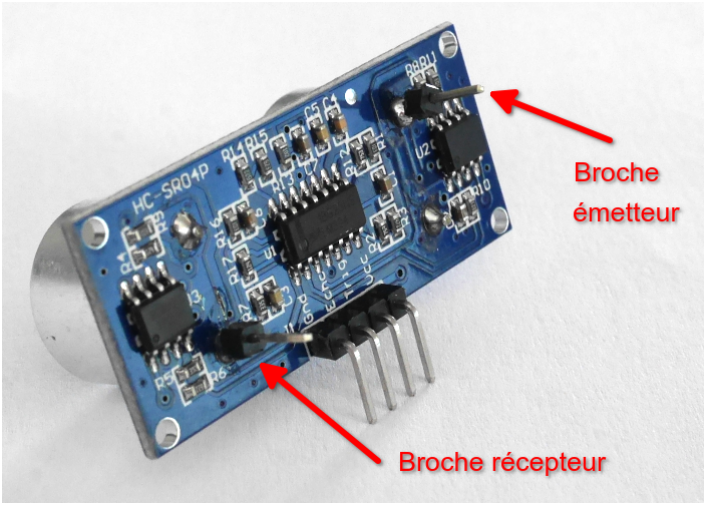



FIG. 11 – Détail du module HC-SR04

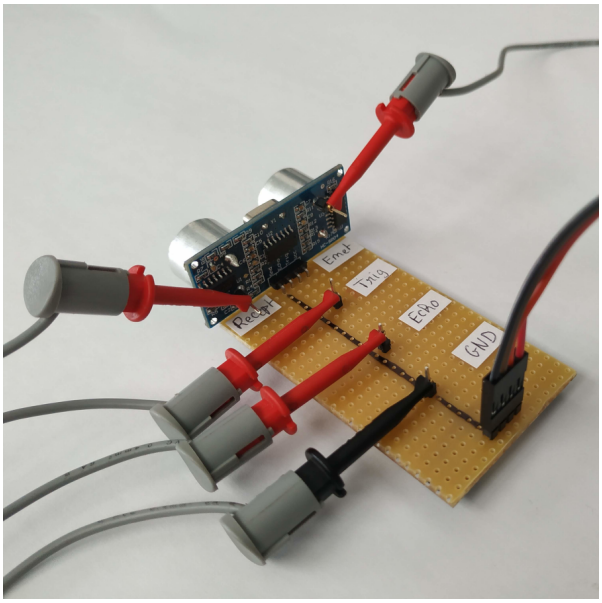



FIG. 12 – Branchement du module HC-SR04 modifié

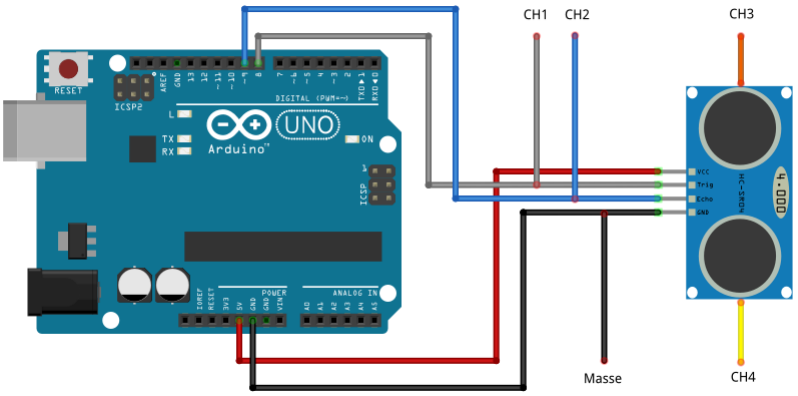



FIG. 13 – Montage avec oscilloscope ou interface d'acquisition

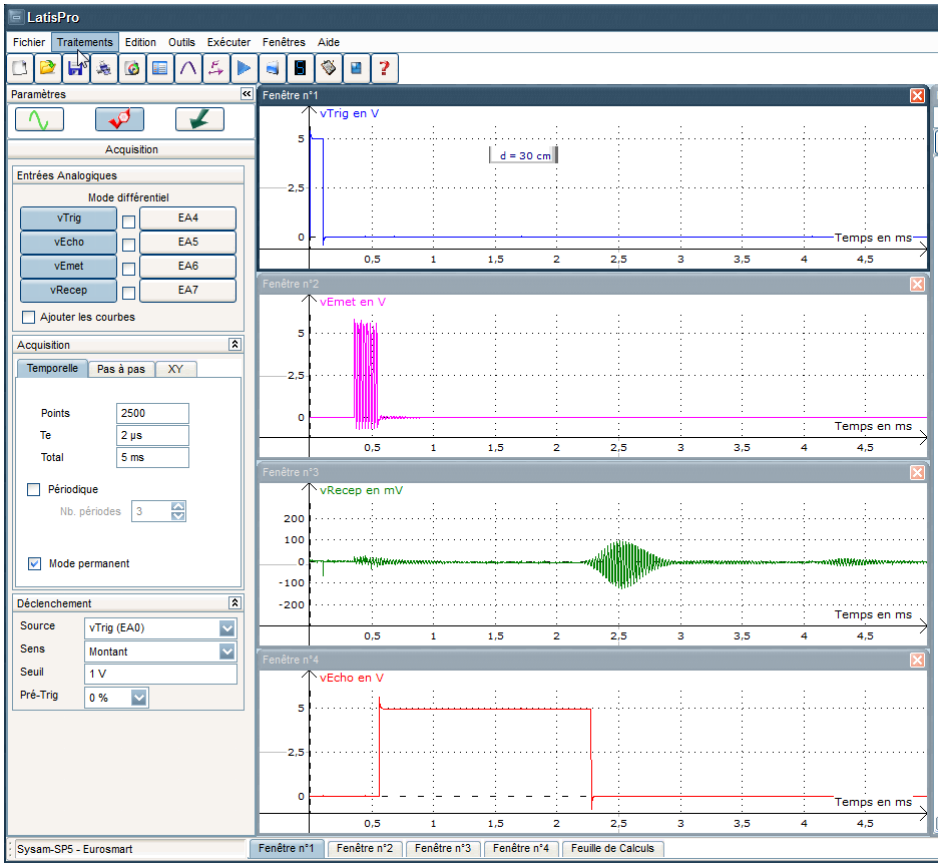



FIG. 14 – Mesures obtenues dans Latis avec Sysam SP5

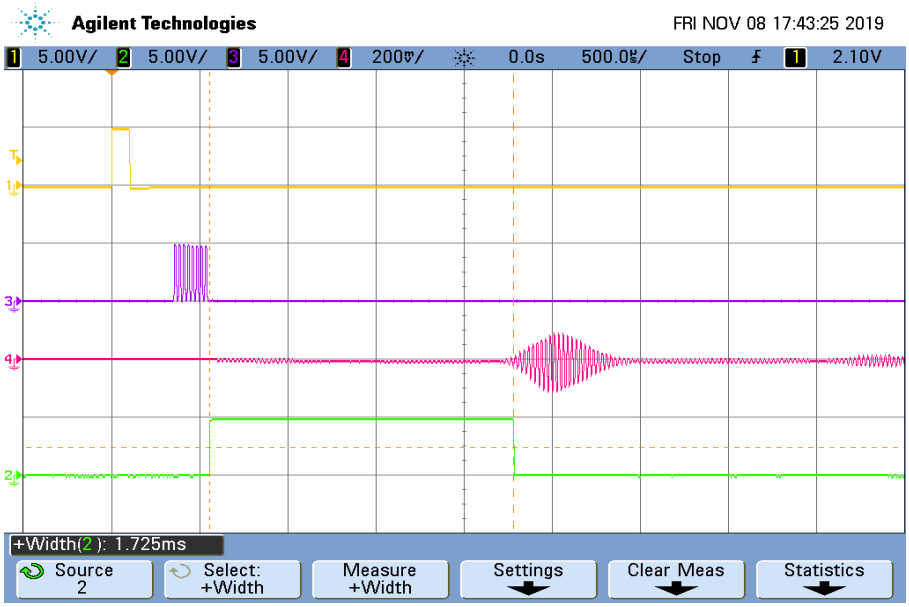



FIG. 15 – Mesures à l'oscilloscope pour une distance de 30 cm

### 6.3.2 Mesure de la célérité du son

La manipulation consiste à relever la durée de l'écho sonore à l'aide du microcontrôleur pour différentes distances et déduire la célérité du son.

$$c = \frac{2 \times d}{\Delta t}$$

#### 6.3.2.1 Algorithme

```
TRIG <- 0

REPETER :
   TRIG <- Vcc                          # Début impulsion sur Trig
   Attendre 10 µs
   TRIG <- 0                            # Fin impulsion sur Trig
   Dt   <- Durée impulsion sur Echo     # Mesure
   Afficher Dt
   Attendre 1 s
```

Pour plus de précision, il est possible de modifier le programme afin de **réaliser plusieurs mesures** de la durée de l'écho et d'en déduire sa **moyenne**.

#### 6.3.2.2 Arduino (C/C++)

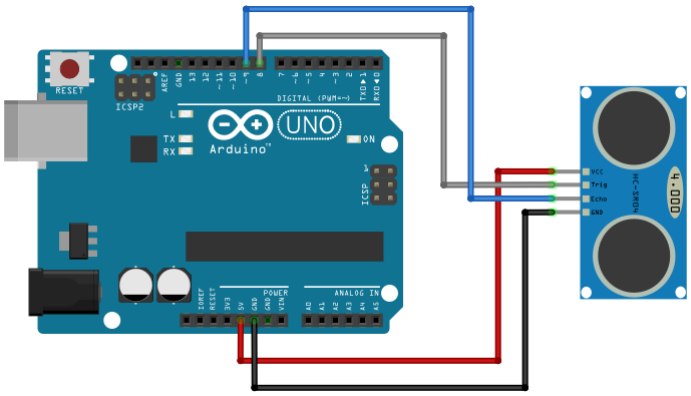


FIG. 16 – Montage célérité son

```
// Mesure de la durée de l'écho sonore

#define pinTrig 8       // Trig sur broche 8
#define pinEcho 9       // Echo sur broche 9

long dureeEcho;         // Durée de l'Echo

void setup() {
  pinMode(pinTrig,OUTPUT);      // Broche Trig en sortie
  digitalWrite(pinEcho,LOW);    // Sortie Trig à l état bas
  pinMode(pinEcho,INPUT);       // Broche Echo en entrée
  Serial.begin(9600);           // Paramétrage du port série
}

void loop() {
  digitalWrite(pinTrig,HIGH);           // Début impulsion de declenchement
  delayMicroseconds(10);                // Attendre 10 microseconde
  digitalWrite(pinTrig,LOW);            // Fin impulsion (Etat bas)
  dureeEcho = pulseIn(pinEcho,HIGH);    // Mesure de la durée de l'impulsion sur Echo
  Serial.print("Durée (µs) = ");         // Affichage sur port série
  Serial.println(dureeEcho);
  delay(1000);                          // Attendre 1s
}
```

#### 6.3.2.3 Arduino (Python/Nanpy)

```
# Réalisé avec une version modifiée de Nanpy par Eurosmart
from nanpy import SerialManager, Ultrasonic
from time import sleep

port = SerialManager(device='/dev/ttyACM0')   # Sélection du port série (exemple :␣
 ↪device = 'COM6')

moduleUltrason = Ultrasonic(trig=8, echo=9, useInches=False, connection=port) #␣
 ↪Déclaration du module HC-SR04
sleep(0.1)                                    # Temporisation
```

(suite sur la page suivante)

(suite de la page précédente)

```
for i in range(10):
    duree = moduleUltrason.get_duration()     # Durée en µs pour l'aller-retour du son
    print("Durée =", duree, "µs")             # Affichage
    sleep(1)                                  # Temporisation
```

Résultats :

```
Durée = 1456 µs
Durée = 1453 µs
Durée = 1451 µs
Durée = 1450 µs
Durée = 1453 µs
Durée = 1453 µs
Durée = 1447 µs
Durée = 1444 µs
Durée = 1452 µs
Durée = 1446 µs
```

### 6.3.3 Application : réalisation d'un télémètre

Connaissant la célérité du son, la distance par rapport à un obstacle est calculée par le microcontrôleur à l'aide de la relation suivante :

$$d = \frac{c \times \Delta t}{2}$$

Il suffit d'ajouter le calcul de la distance juste après la mesure de la durée.

#### 6.3.3.1 Arduino (C/C++)

```
// Mesure de la durée d'une distance

#define pinTrig 8       // Trig sur broche 8
#define pinEcho 9       // Echo sur broche 9

long dureeEcho;         // Durée de l'Echo
float distance;         // Distance en module et réflecteur
float vitesse = 345 ;   // Vitesse obtenue

void setup() {
pinMode(pinTrig,OUTPUT);      // Broche Trig en sortie
digitalWrite(pinEcho,LOW);    // Sortie Trig à l état bas
pinMode(pinEcho,INPUT);       // Broche Echo en entrée
Serial.begin(9600);           // Paramétrage du port série
}

void loop() {
digitalWrite(pinTrig,HIGH);            // Début impulsion de declenchement
delayMicroseconds(10);                 // Attendre 10 microseconde
digitalWrite(pinTrig,LOW);             // Fin impulsion (Etat bas)
dureeEcho = pulseIn(pinEcho,HIGH);     // Mesure de la durée de l'impulsion sur Echo
distance = (vitesse*dureeEcho*1E-6)/2; // Calcul de la distance
Serial.print("Duree (us) = ");         // Affichage sur port série
Serial.println(dureeEcho);             //
```

(suite sur la page suivante)

(suite de la page précédente)

```
Serial.print("Distance (m) = ");        // Affichage sur port série
Serial.println(distance);               //
delay(1000);                            // Attendre 1s
}
```

En autonome avec un écran LCD :

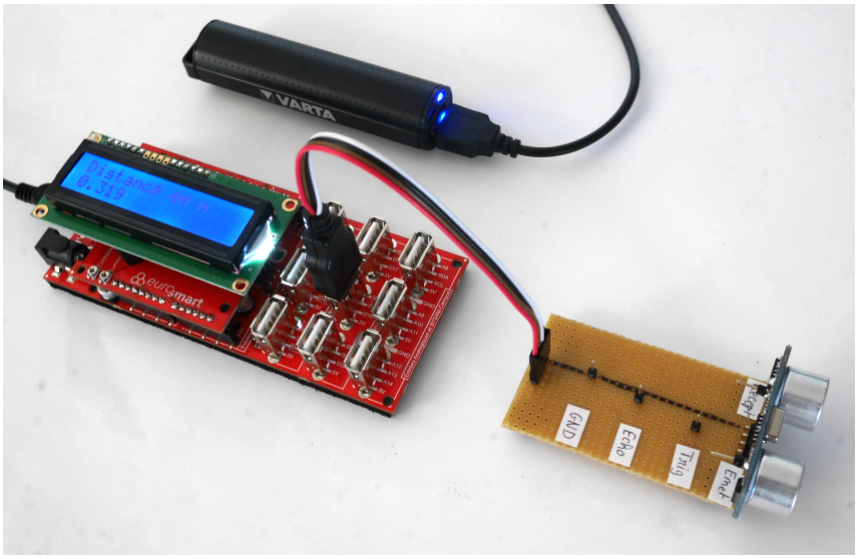



FIG. 17 – Télémètre sur Educaduino-Lab LCD

```
//Application : télémètre sur écran LCD 2x16

#include <LiquidCrystal.h>        // Importation de la librairie LiquidCrystal
#define pinTrig 8       // Trig sur broche 8
#define pinEcho 9       // Echo sur broche 9

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // Brochage de l'afficheur

float distance;         // Distance en module et réflecteur
long dureeEcho;         // Durée mesurée
float vitesse = 345 ;   // Vitesse obtenue


void setup() {
 pinMode(pinTrig,OUTPUT);       // Broche Trig en sortie
 digitalWrite(pinEcho,LOW);     // Sortie Trig à l état bas
 pinMode(pinEcho,INPUT);        // Broche Echo en entrée
 lcd.begin(16, 2);                // fixe le nombre de colonnes et de lignes de l␣
 ↪afficheur
}

void loop() {
 digitalWrite(pinTrig,HIGH);                    // Début impulsion de déclenchement
 delayMicroseconds(10);                         // Attendre 10 microseconde
 digitalWrite(pinTrig,LOW);                     // Fin impulsion (Etat bas)
 dureeEcho = pulseIn(pinEcho,HIGH);             // Mesure de la durée de l'impulsion sur␣
 ↪Echo
 distance = (vitesse*dureeEcho*1E-6)/2;     // Calcul de la distance
```

(suite sur la page suivante)

(suite de la page précédente)

```
 lcd.setCursor(0,0);                              // place le curseur au début de la ligne␣
 ↪0
 lcd.print("Distance en m");                     // Affiche la légende
 lcd.setCursor(0,1);                              // place le curseur au début de la ligne␣
 ↪1
 lcd.print(distance);                             // Affiche la valeur de la distance
 delay(1000);                                     // Attendre 1s
}
```

#### 6.3.3.2 Arduino (Python/Nanpy)

```
# Réalisé avec une version modifiée de Nanpy par Eurosmart
from nanpy import SerialManager, Ultrasonic
from time import sleep

port = SerialManager(device='/dev/ttyACM0')   # Sélection du port série (exemple :␣
 ↪device = 'COM6')

moduleUltrason = Ultrasonic(trig=8, echo=9, useInches=False, connection=port) #␣
 ↪Déclaration du module HC-SR04
sleep(0.1)                                     # Temporisation

vitesse_son = 345                              # vitesse du son 345 m/S dans l'air

for i in range(10):
   duree = moduleUltrason.get_duration()     # Durée en µs pour l'aller-retour du son
   print("Durée =", duree, "µs")             # Affichage
   distance = (vitesse_son*duree*1E-6)/2;    # Calcul de la distance en m
   print("Distance = ", distance, "m")       # Affichage
   sleep(1)                                  # Temporisation
```

Résultats :

```
Durée = 1997 µs
Distance =  0.34448249999999997 m
Durée = 1996 µs
Distance =  0.34431 m
Durée = 2019 µs
Distance =  0.34827749999999996 m
Durée = 8214 µs
Distance =  1.416915 m
Durée = 8181 µs
Distance =  1.4112225 m
Durée = 8177 µs
Distance =  1.4105325 m
Durée = 1822 µs
Distance =  0.314295 m
Durée = 1915 µs
Distance =  0.3303375 m
```

### 6.3.4 A retenir

— Le module HC-SR04 fournit un **signal Echo pour la mesure automatique de la durée** de propagation du son.
— La fonction `pulseIn()` **mesure cette durée**.

## 6.4 Mesurer une pression - Loi de Mariotte (première générale)

**Programme de première générale 2019 - Enseignement de spécialité.**

Tester la loi de Mariotte, par exemple en utilisant un dispositif comportant un microcontrôleur.

### 6.4.1 Principe

La manipulation consiste à vérifier la loi de Mariotte $P \times V = constante$ (à température et quantité de matière constantes).

### 6.4.2 Capteur de pression absolue MPX5700AP

Le MPX5700AP est un capteur analogique de pression absolue (15 kPa à 700 kPa - 5V).

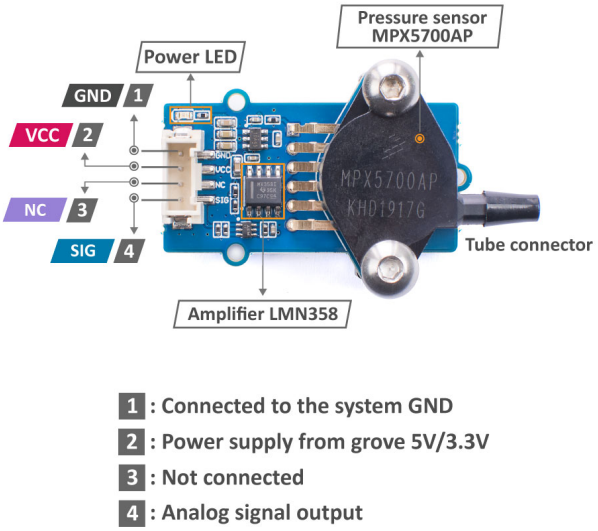



FIG. 18 – Capteur MPX5700AP Grove (15 kPa à 700 kPa) (image : seeedstudio)

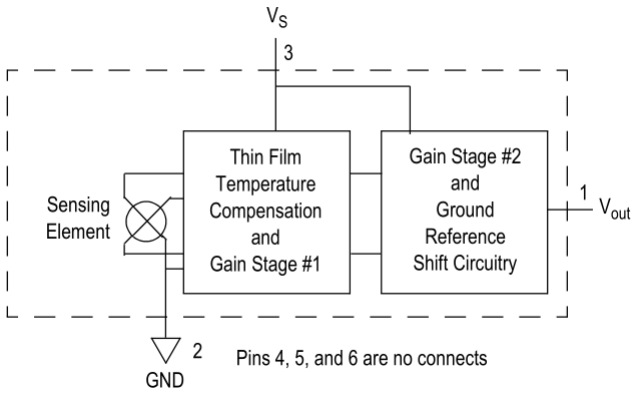



FIG. 19 – Schéma électrique (source : Freescale Semiconductor, Inc.)

**Table 1. Operating Characteristics** ($V_S$ = 5.0 Vdc, $T_A$ = 25°C unless otherwise noted, P1 > P2. Decoupling circuit shown in required to meet electrical specifications.)

| Characteristic | | Symbol | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Pressure Range(1) | Gauge, Differential: MPX5700D<br>Absolute: MPX5700A | $P_{OP}$ | 0<br>15 | —<br>— | 700<br>700 | kPa |
| Supply Voltage(2) | | $V_S$ | 4.75 | 5.0 | 5.25 | Vdc |
| Supply Current | | $I_O$ | — | 7.0 | 10 | mAdc |
| Zero Pressure Offset(3) | Gauge, Differential (0 to 85°C)<br>Absolute (0 to 85°C) | $V_{off}$ | 0.088<br>0.184 | 0.2<br>— | 0.313<br>0.409 | Vdc |
| Full Scale Output(4) | (0 to 85°C) | $V_{FSO}$ | 4.587 | 4.7 | 4.813 | Vdc |
| Full Scale Span(5) | (0 to 85°C) | $V_{FSS}$ | — | 4.5 | — | Vdc |
| Accuracy(6) | (0 to 85°C) | — | — | — | ±2.5 | %$V_{FSS}$ |
| Sensitivity | | V/P | — | 6.4 | — | mV/kPa |
| Response Time(7) | | $t_R$ | — | 1.0 | — | ms |
| Output Source Current at Full Scale Output | | $I_{O+}$ | — | 0.1 | — | mAdc |
| Warm-Up Time(8) | | — | — | 20 | — | ms |

FIG. 20 – Caractéristiques (source : Freescale Semiconductor, Inc.)

La mesure de pression (en kPa) est donnée par la relation pour une alimentation de 5 V :

$$P = \frac{P_{max} - P_{min}}{V_{max} - V_{min}} \times (v_{out} - V_{min}) + P_{min}$$

Soit :

$$P = \frac{700 - 15}{4,7 - 0,2} \times (v_{out} - 0,2) + 15 \quad \Longrightarrow \quad P \approx 152 \times (v_{out} - 0,2) + 15$$

### 6.4.3 Capteur de pression absolu MPXHZ6400A (Educaduino LAB)

Le MPXHZ6400A est un capteur analogique de pression absolue (20 kPa à 400 kPa - 5V).


FIG. 21 – Capteur MPX6400A Educaduino LAB

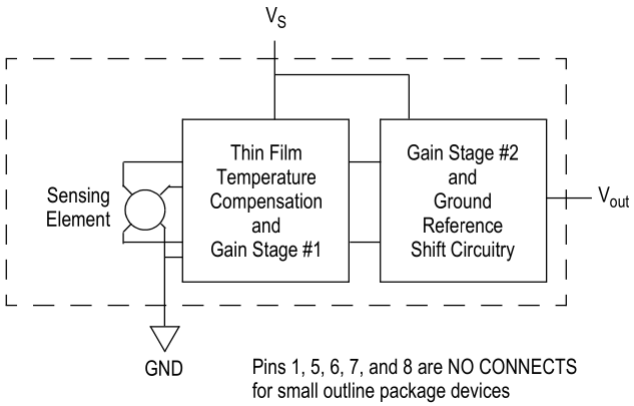



FIG. 22 – Schéma électrique (source : Freescale Semiconductor, Inc.)

**Table 2. Operating Characteristics** ($V_S$ = 5.0 Vdc, $T_A$ = 25°C.)

| Characteristic | | Symbol | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| Pressure Range | | $P_{OP}$ | 20 | — | 400 | kPa |
| Supply Voltage(1) | | $V_S$ | 4.64 | 5.0 | 5.36 | Vdc |
| Supply Current | | $I_o$ | — | 6.0 | 10 | mAdc |
| Minimum Pressure Offset @ $V_S$ = 5.0 Volts(2) | (0 to 85°C) | $V_{off}$ | 0.133 | 0.2 | 0.267 | Vdc |
| Full Scale Output @ $V_S$ = 5.0 Volts(3) | (0 to 85°C) | $V_{FSO}$ | 4.733 | 4.8 | 4.866 | Vdc |
| Full Scale Span @ $V_S$ = 5.0 Volts(4) | (0 to 85°C) | $V_{FSS}$ | 4.467 | 4.6 | 4.733 | Vdc |
| Accuracy(5) | (0 to 85°C) | — | — | — | ±1.5 | $\%V_{FSS}$ |
| Sensitivity | | V/P | — | 12.1 | — | mV/kPa |
| Response Time(6) | | $t_R$ | — | 1.0 | — | ms |
| Warm-Up Time(7) | | — | — | 20 | — | ms |
| Offset Stability(8) | | — | — | ±0.25 | — | $\%V_{FSS}$ |

FIG. 23 – Caractéristiques (source : Freescale Semiconductor, Inc.)

La mesure de pression (en kPa) est donnée par la relation pour une alimentation de 5 V :

$$P = \frac{P_{max} - P_{min}}{V_{max} - V_{min}} \times (v_{out} - V_{min}) + P_{min}$$

Soit :

$$P = \frac{400 - 20}{4,8 - 0,2} \times (v_{out} - 0,2) + 20 \quad \Longrightarrow \quad P \approx 82,6 \times (v_{out} - 0,2) + 20$$

### 6.4.4 Arduino (C/C++)

Le montage est composé d'une carte Educaduino Lab, d'un capteur de pression Educaduino (MPXHZ6400A / 20 kPa à 400 kPa) et d'une seringue.

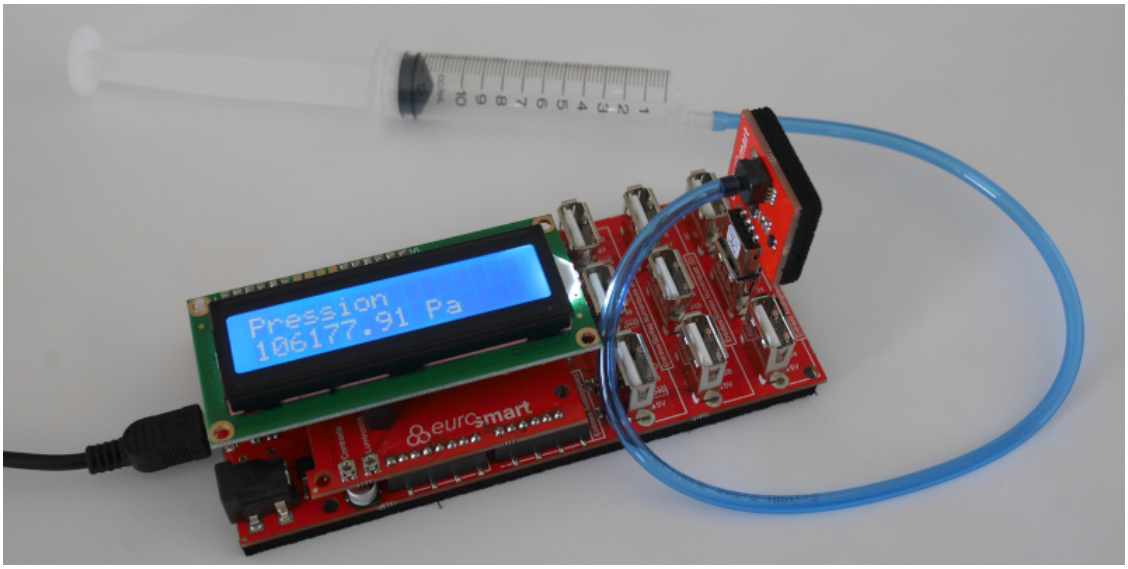



FIG. 24 – Mesure d'une pression avec Educaduino Lab

**Note**

Dans cette manipulation, il est important de tenir compte du volume d'air $V_0$ présent dans le tube. La loi de Mariotte s'écrit alors

$$P \times (V + V_0) = constante$$

```
/*
 * Mesure d'une pression absolue
 * Capteur Educaduino 20 kPa à 400 kPa
 * branché sur la broche A9
 */

#define brocheCapteur A9      // Numéro de broche connectée au capteur
#include <LiquidCrystal.h>    // Librairie de gestion de l écran LCD

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // Déclaration de l écran LCD


float tension ;               // Tension mesurée
float pression ;              // Pression mesurée

void setup() {
  lcd.begin(16, 2);           // Paramétrage de l ecran LCD

}

void loop() {
  tension = analogRead(brocheCapteur)*5.0/1023 ;   // Lecture de la tension
  pression = 82.6*(tension-0.2) + 20 ;               // Calcul de la pression en kPa
  lcd.clear();                                      // Début affichage
  lcd.setCursor(0,0);
  lcd.print("Pression en kPa");
  lcd.setCursor(0,1);
```

(suite sur la page suivante)

(suite de la page précédente)

```
  lcd.print(pression);                                  // Fin affichage
  delay(1000);
}
```

Résultats :

| V (mL) | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |
|---|---|---|---|---|---|---|---|---|---|
| P (kPa) | 96,5 | 105 | 114 | 124 | 136 | 152 | 171 | 194 | 227 |

### 6.4.5 Arduino (Python/Nanpy)

Cet exemple utilise un module Grove MPX5700AP (15-700 kPa). Les mesures sont affichées au format CSV pour exploitation avec un tableur, Regressi, Latis ou Python par un copier-coller.

```
# Vérification de la loi de Boyle-Mariotte avec module Grove MPX5700AP 15-700 kPa > 0.2-
 ↪4.7 V
from nanpy import ArduinoApi                    # Gestion de la carte Arduino
from nanpy import SerialManager                 # Gestion du port série

port = SerialManager(device='/dev/ttyACM0')     # Sélection du port série (exemple :␣
 ↪device = 'COM6')
uno = ArduinoApi(connection=port)               # Déclaration de la carte Arduino

Pmin = 15     # Pression minimale en kPa
Pmax = 700    # Pression maximale en kPa
Umin = 41     # Tension minimale 0.2/5*1023 = 41
Umax = 962    # Tension maximale 4.7/5*1023 = 962

volume   = [60,50,40,35,30,25]    # Proposition de volumes - 40 mL pour pression␣
 ↪atmosphérique
pression = []                     # Tableau des pressions

# Mesures
for vol in volume :                                      # Parcours des volumes prédéfinis
   input("Régler le volume sur " + str(vol) + " mL") # Validation du réglage du volume
   U = uno.analogRead(0)                                 # Lecture de la tension numérique␣
 ↪(10 bit)
   P = (Pmax-Pmin)/(Umax-Umin)*(U-Umin) + Pmin           # Calcul de la pression
   print(P, "kPa")                                       # Affichage de la pression
   pression.append(P)                                    # Ajout de la mesure dans le␣
 ↪tableau de pression

# Affichage au format CSV
print("V ; P")                          # Affichage entête des grandeurs
print("mL ; hPa")                       # Affichage entête des unités
for i in range(len(volume)):            # Parcours des points de mesures
   print(volume[i],";",pression[i])  # Affichage des mesures
```

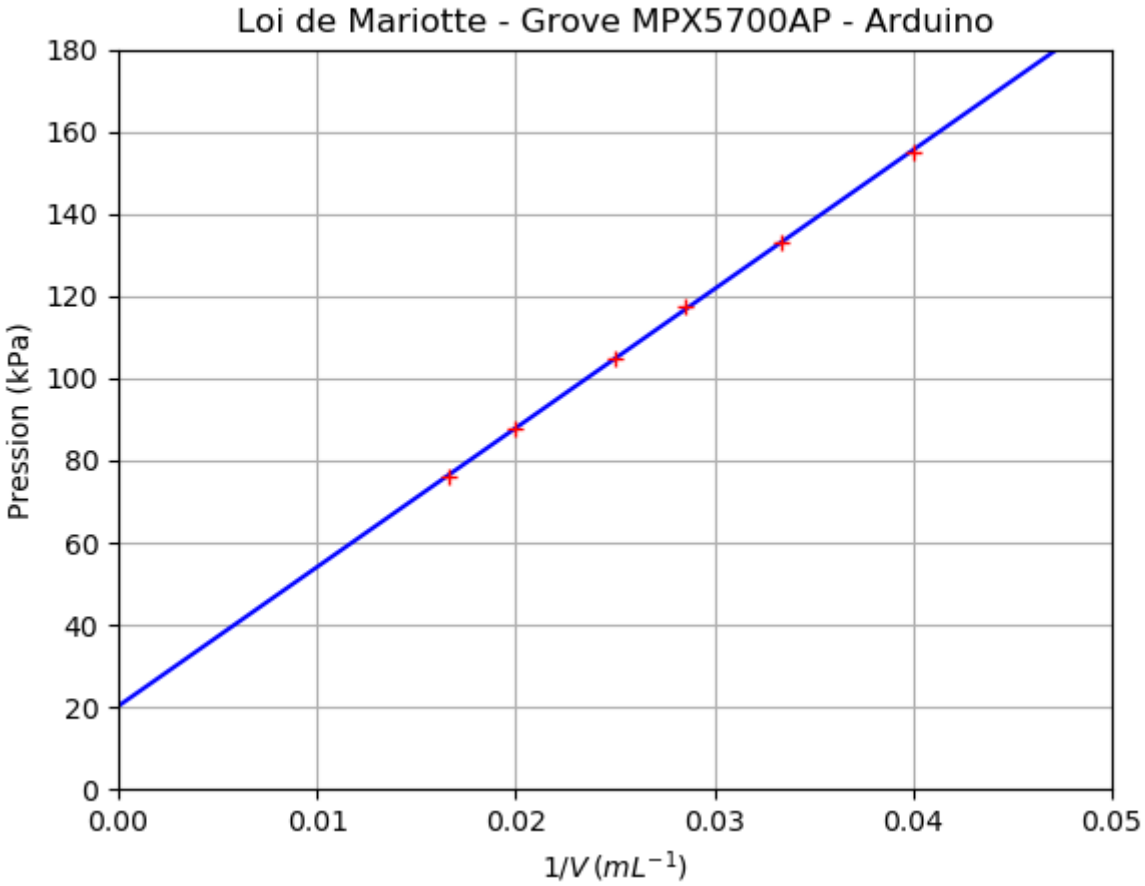
Résultats :

```
Régler le volume sur 60 mL
75.98805646036917 kPa
Régler le volume sur 50 mL
87.88816503800217 kPa
Régler le volume sur 40 mL
```

(suite sur la page suivante)

(suite de la page précédente)

```
104.99457111834963 kPa
Régler le volume sur 35 mL
117.63843648208469 kPa
Régler le volume sur 30 mL
133.25732899022802 kPa
Régler le volume sur 25 mL
154.82627578718783 kPa

V ; P
mL ; kPa
60 ; 75.98805646036917
50 ; 87.88816503800217
40 ; 104.99457111834963
35 ; 117.63843648208469
30 ; 133.25732899022802
25 ; 154.82627578718783
```




### 6.4.6 Pyboard (Micropython)

Cet exemple utilise également un module Grove MPX5700AP (15-700 kPa). Les mesures sont affichées au format CSV pour exploitation avec un tableur, Regressi, Latis ou Python par un copier-coller.

```
# Vérification de la loi de Boyle-Mariotte avec module Grove MPX5700AP 15-700 kPa > 0.2-
↪4.7 V
```

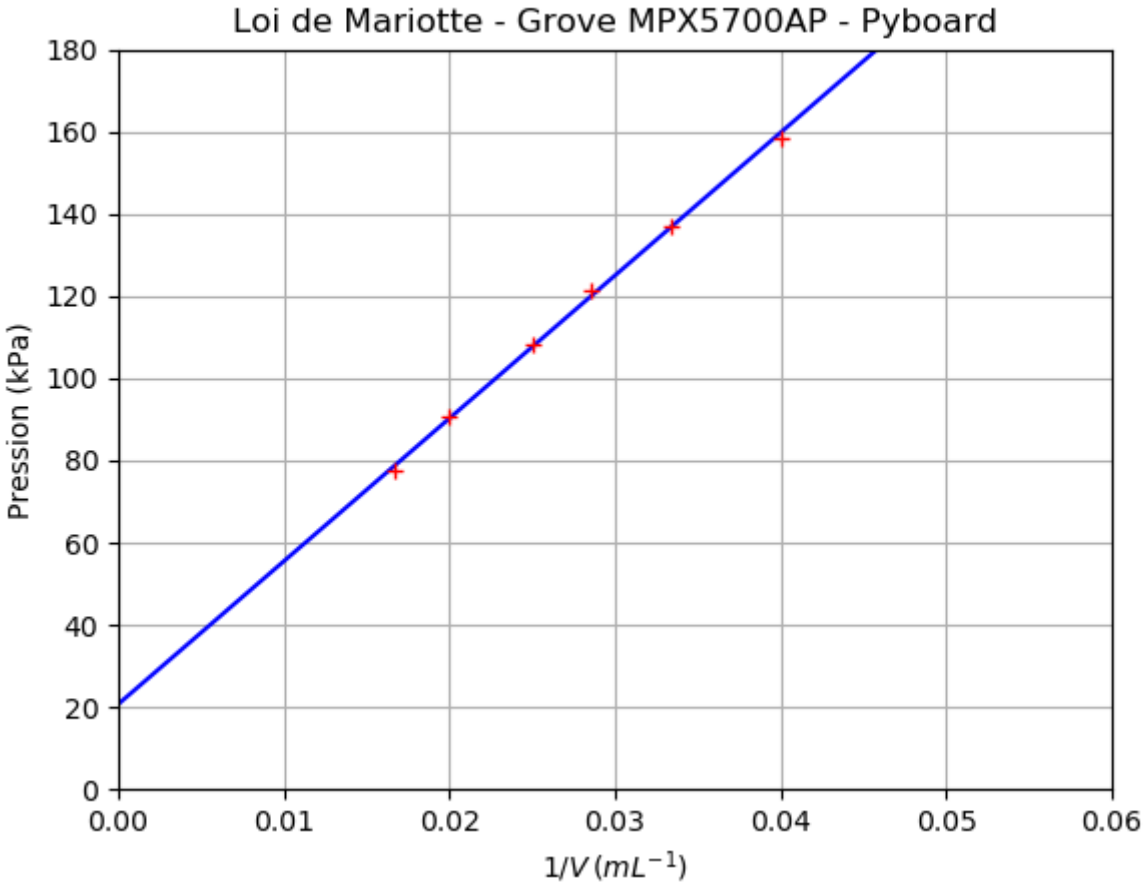
(suite sur la page suivante)

(suite de la page précédente)

```
from pyb import Pin, ADC

adc = ADC(Pin("A0"))                # Déclaration du CAN

Pmin = 15                           # Pression minimale
Pmax = 700                          # Pression maximale
                                    # Amplification = 3,3/4,7 = 0.702
Umin = 174                          # Tension minimale (0.2V * 0.702 = 0.140V) N=174
Umax = 4095                         # Tension maximale (4.7V * 0.702 = 3.300V) N=4095

volume   = [60,50,40,35,30,25]      # Proposition de volumes - 40 mL pour pression␣
 ↪atmosphérique
pression = []                       # Tableau des pressions

# Mesures
for vol in volume :                                    # Parcours des volumes prédéfinis
   input("Régler le volume sur " + str(vol) + " mL") # Validation du réglage du volume
   U = adc.read()                                     # Lecture de la tension numérique␣
 ↪(12 bit)
   P = (Pmax-Pmin)/(Umax-Umin)*(U-Umin) + Pmin        # Calcul de la pression du capteur
   print(P, "kPa")                                    # Affichage de la pression
   pression.append(P)                                 # Ajout de la mesure dans le␣
 ↪tableau de pression

# Affichage au format CSV
print("V ; P")                          # Affichage entête des grandeurs
print("mL ; hPa")                       # Affichage entête des unités
for i in range(len(volume)):            # Parcours des points de mesures
   print(volume[i],";",pression[i])  # Affichage des mesures
```

Résultats :

```
Régler le volume sur 60 mL
79.81383 kPa
Régler le volume sur 50 mL
91.69345 kPa
Régler le volume sur 40 mL
109.1635 kPa
Régler le volume sur 35 mL
122.0913 kPa
Régler le volume sur 30 mL
137.4649 kPa
Régler le volume sur 25 mL
157.7302 kPa

V ; P
mL ; kPa
60 ; 79.81383
50 ; 91.69345
40 ; 109.1635
35 ; 122.0913
30 ; 137.4649
25 ; 157.7302
```




### 6.4.7 Micro :bit (Micropython)

Cet exemple utilise également un module Grove MPX5700AP (15-700 kPa). Les mesures sont affichées au format CSV pour exploitation avec un tableur, Regressi, Latis ou Python par un copier-coller.

```
# Vérification de la loi de Boyle-Mariotte avec module Grove MPX5700AP 15-700 kPa > 0.2-
 ↪4.7 V
from microbit import *

Vcc = 3.09     # Mesure au voltmètre entre 3V3 et GND
No  = 4        # Décalage (entier) obtenu pour une tension de 0V

Pmin = 15      # Pression minimale
Pmax = 700     # Pression maximale
            # Amplification = 3,3/4,7 = 0.702
Umin = 0.14    # Tension minimale (0.2V * 0.702 = 0.140V) N=44
Umax = 3.3     # Tension maximale (4.7V * 0.702 = 3.300V) N=1023

volume   = [60,50,40,35,30,25]     # Proposition de volumes - 40 mL pour pression␣
 ↪atmosphérique
pression = []                      # Tableau des pressions

# Mesures
for vol in volume :                                    # Parcours des volumes prédéfinis
   input("Régler le volume sur " + str(vol) + " mL") # Validation du réglage du volume
```

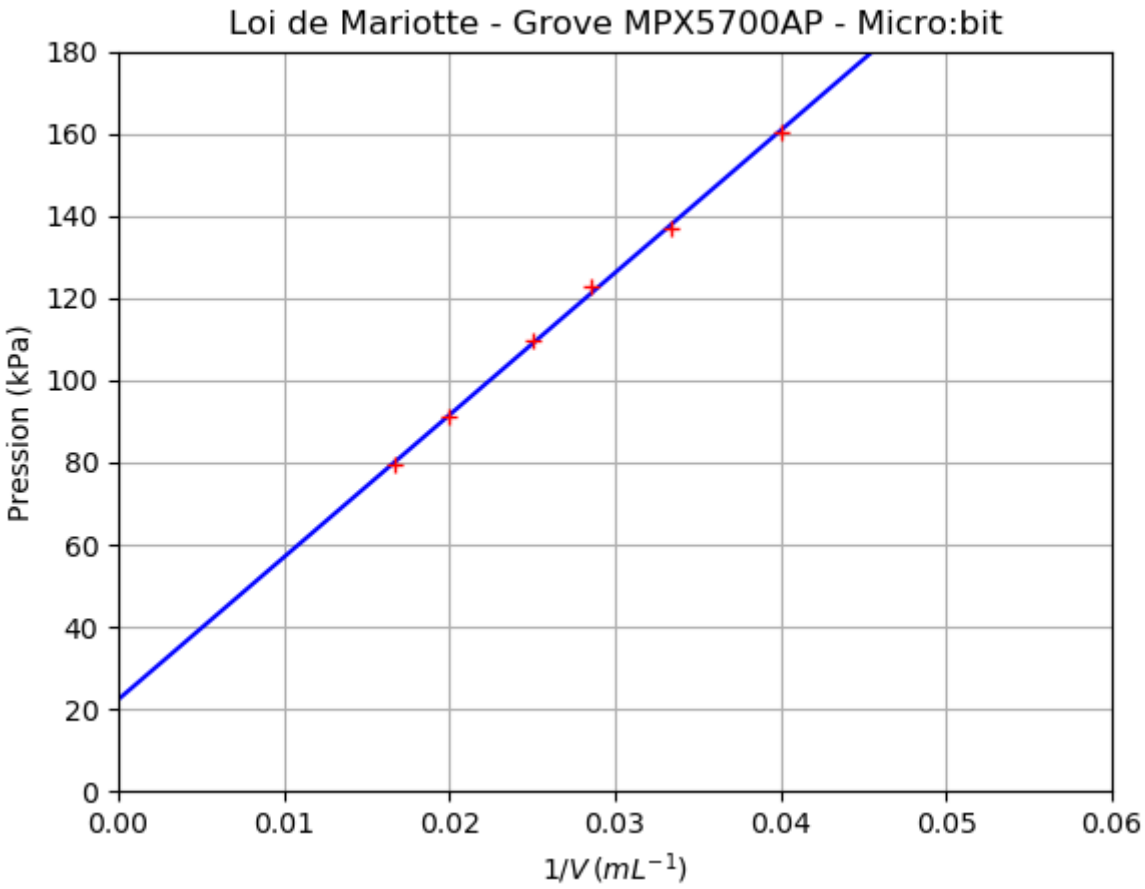
(suite sur la page suivante)

(suite de la page précédente)

```
    N = pin1.read_analog()                                  # Lecture de la tension numérique␣
 ↪(10 bit)
    U = (N-No)*Vcc/(1023-No)                                # Calcul de la tension
    P = (Pmax-Pmin)/(Umax-Umin)*(U-Umin) + Pmin             # Calcul de la pression
    print(P, "kPa")                                         # Affichage de la pression
    pression.append(P)                                      # Ajout de la mesure dans le␣
 ↪tableau de pression

# Affichage au format CSV
print("V ; P")                          # Affichage entête des grandeurs
print("mL ; hPa")                       # Affichage entête des unités
for i in range(len(volume)):            # Parcours des points de mesures
    print(volume[i],";",pression[i])  # Affichage des mesures
```

Résultats :

```
Régler le volume sur 60 mL
79.3083 kPa
Régler le volume sur 50 mL
91.1404 kPa
Régler le volume sur 40 mL
109.546 kPa
Régler le volume sur 35 mL
122.693 kPa
Régler le volume sur 30 mL
137.154 kPa
Régler le volume sur 25 mL
160.161 kPa

V ; P
mL ; kPa
60 ; 79.3083
50 ; 91.1404
40 ; 109.546
35 ; 122.693
30 ; 137.154
25 ; 160.161
```




## 6.5 Mesurer une pression - Loi de la statique des fluides (première générale)

**Programme de première générale 2019 - Enseignement de spécialité.**

Tester la loi fondamentale de la statique des fluides.

### 6.5.1 Capteur MPX2010DP/GP

La mesure de pression s'effectue avec un capteur de pression différentiel du type MPX2010 (0 à 100 hPa) compensé en température.

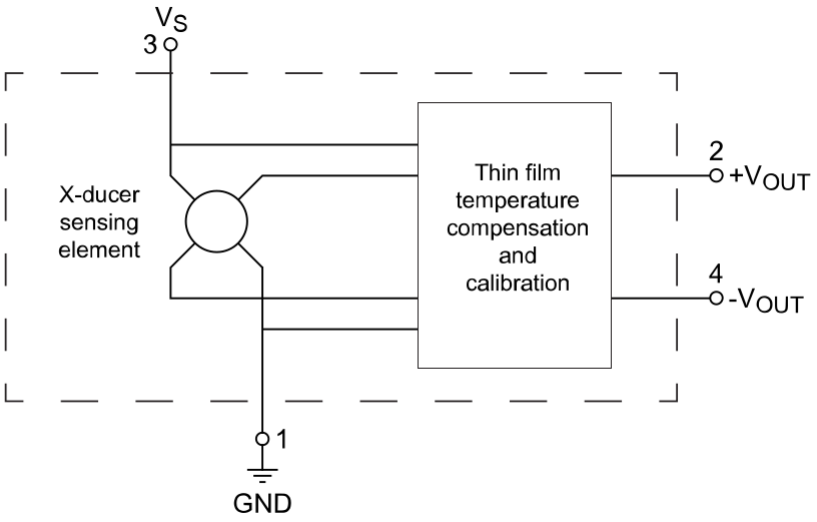



FIG. 25 – Schéma électrique du MPX2010 (image : NXP Semiconductors)

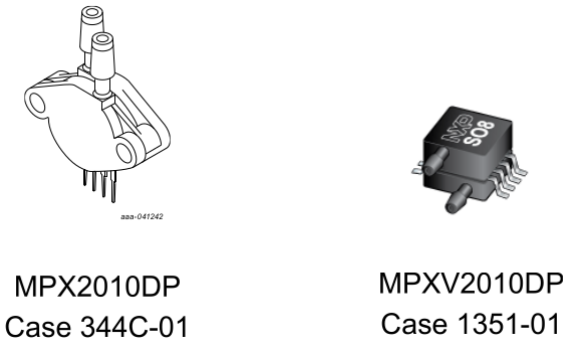



FIG. 26 – (image : NXP Semiconductors)

La **version DP** (pression différentielle) mesure la différence de pression entre deux points d'entrée.


FIG. 27 – (image : NXP Semiconductors)

La **version GP** (pression relative - manomètre) mesure la pression d'un point d'entrée par rapport à une pression de référence (pression atmosphérique).

Table 9. Operating characteristics ($V_S$ = 10 Vdc, $T_A$ = 25 °C unless otherwise noted, P1 > P2)

| Characteristic | | Symbol | Min | Typ | Max | Units |
|---|---|---|---|---|---|---|
| Operating Pressure Range | [1] | $P_{OP}$ | 0 | — | 10 | kPa |
| Supply Voltage | [2] | $V_S$ | — | 10 | 16 | Vdc |
| Supply Current | | $I_o$ | — | 6.0 | — | mAdc |
| Full Scale Span | [3] | $V_{FSS}$ | 24 | 25 | 26 | mV |
| Offset | [4] | $V_{off}$ | –1.0 | — | 1.0 | mV |
| Sensitivity | | ΔV/ΔP | — | 2.5 | — | mV/kPa |
| Linearity | [5] | — | –1.0 | — | 1.0 | $\%V_{FSS}$ |
| Pressure Hysteresis (0 kPa to 10 kPa) | [5] | — | — | ±0.1 | — | $\%V_{FSS}$ |
| Temperature Hysteresis (–40 °C to +125 °C) | [5] | — | — | ±0.5 | — | $\%V_{FSS}$ |
| Temperature Coefficient of Full Scale Span | [5] | $TCV_{FSS}$ | –1.0 | — | 1.0 | $\%V_{FSS}$ |
| Temperature Coefficient of Offset | [5] | $TCV_{off}$ | –1.0 | — | 1.0 | mV |
| Input Impedance | | $Z_{in}$ | 1300 | — | 2550 | Ω |
| Output Impedance | | $Z_{out}$ | 1400 | — | 3000 | Ω |
| Response Time (10% to 90%) | [6] | $t_R$ | — | 1.0 | — | ms |
| Warm-Up Time | [7] | — | — | 20 | — | ms |
| Offset Stability | [8] | — | — | ±0.5 | — | $\%V_{FSS}$ |

FIG. 28 – Extrait datasheet (source : NXP Semiconductors)

Pour ce capteur, la tension de sortie est proportionnelle à la différence de pression mesurée telle que :

$$U_{out} = S \times \Delta P \qquad \text{avec} \qquad S = 2,5 \text{ mV/kPa}$$

### 6.5.2 Module Educaduino Lab (MPX2010GP)

Le module Educaduino LAB est conçu autour sur le capteur MPX2010GP.


FIG. 29 – Capteur de pression différentielle Educaduino LAB (jauge)

Après adaptation (amplification), l'expression de la pression différentielle (en Pa) en fonction de la tension en sortie du

capteur est :

$$\Delta P = 2000 \times U \qquad \text{(Pa)}$$

### 6.5.3 Montage

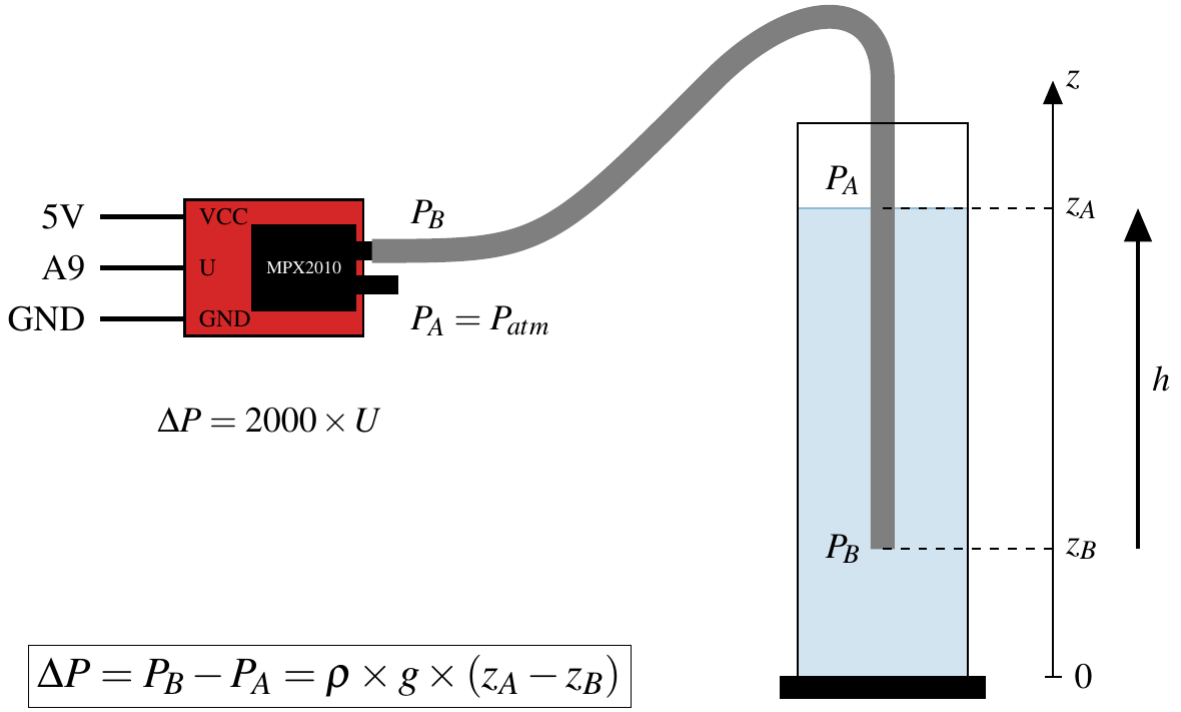



FIG. 30 – Montage de la vérification de loi de la statique des fluides à partir d'un Arduino

### 6.5.4 Programme

Avec écran LCD Educaduino LAB. La lecture de la tension analogique se fait sur la broche A9.

```
/*
 * Mesure d une pression relative
 * Capteur Educaduino MPX2010GP 0 à 10 kPa
 * branché sur la broche A9
 */

#define brocheCapteur A9      // Numéro de broche connectée au capteur
#include <LiquidCrystal.h>    // Librairie de gestion de l écran LCD

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // Déclaration de l écran LCD

int n;
float tension ;               // Tension mesurée
int pression ;                // Pression mesurée

void setup() {
  lcd.begin(16, 2);           // Paramétrage de l ecran LCD
```

(suite sur la page suivante)

(suite de la page précédente)

```
}

void loop() {
  n = analogRead(brocheCapteur) ;   // Lecture de la tension
  tension = n*5.0/1023 ;            // Lecture de la tension
  pression = round(tension*2000) ;  // Calcul de la pression en Pa
  lcd.clear();                      // Début affichage
  lcd.setCursor(0,0);
  lcd.print("N");
  lcd.setCursor(6,0);
  lcd.print("P (Pa)");
  lcd.setCursor(0,1);
  lcd.print(n);                     // Fin affichage
  lcd.setCursor(6,1);
  lcd.print(round(pression));       // Fin affichage
  delay(1000);
}
```

## 6.6 Géométrie d'un condensateur (terminale générale)

**Programme de terminale générale 2019 - Enseignement de spécialité.**

Identifier et tester le comportement capacitif d'un dipôle. Illustrer qualitativement, par exemple à l'aide d'un microcontrôleur, d'un multimètre ou d'une carte d'acquisition, **l'effet de la géométrie d'un condensateur sur la valeur de sa capacité**.

### 6.6.1 Principe

### 6.6.2 Montage

### 6.6.3 Programme

### 6.6.4 A retenir

## 6.7 Capteur capacitif (terminale générale)

**Programme de terminale générale 2019 - Enseignement de spécialité.**

Expliquer le principe de fonctionnement de quelques capteurs capacitifs. Étudier la réponse d'un dispositif modélisé par un dipôle RC. **Déterminer le temps caractéristique d'un dipôle RC** à l'aide d'un microcontrôleur, d'une carte d'acquisition ou d'un oscilloscope.

### 6.7.1 Principe

Soit le circuit RC série suivant :

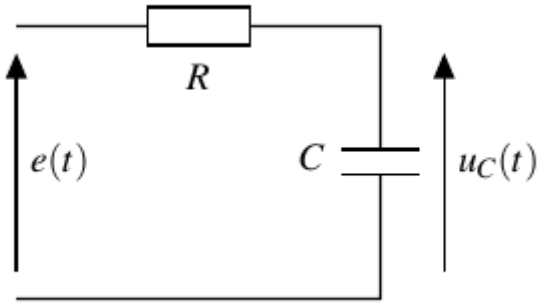



FIG. 31 – Schéma électrique

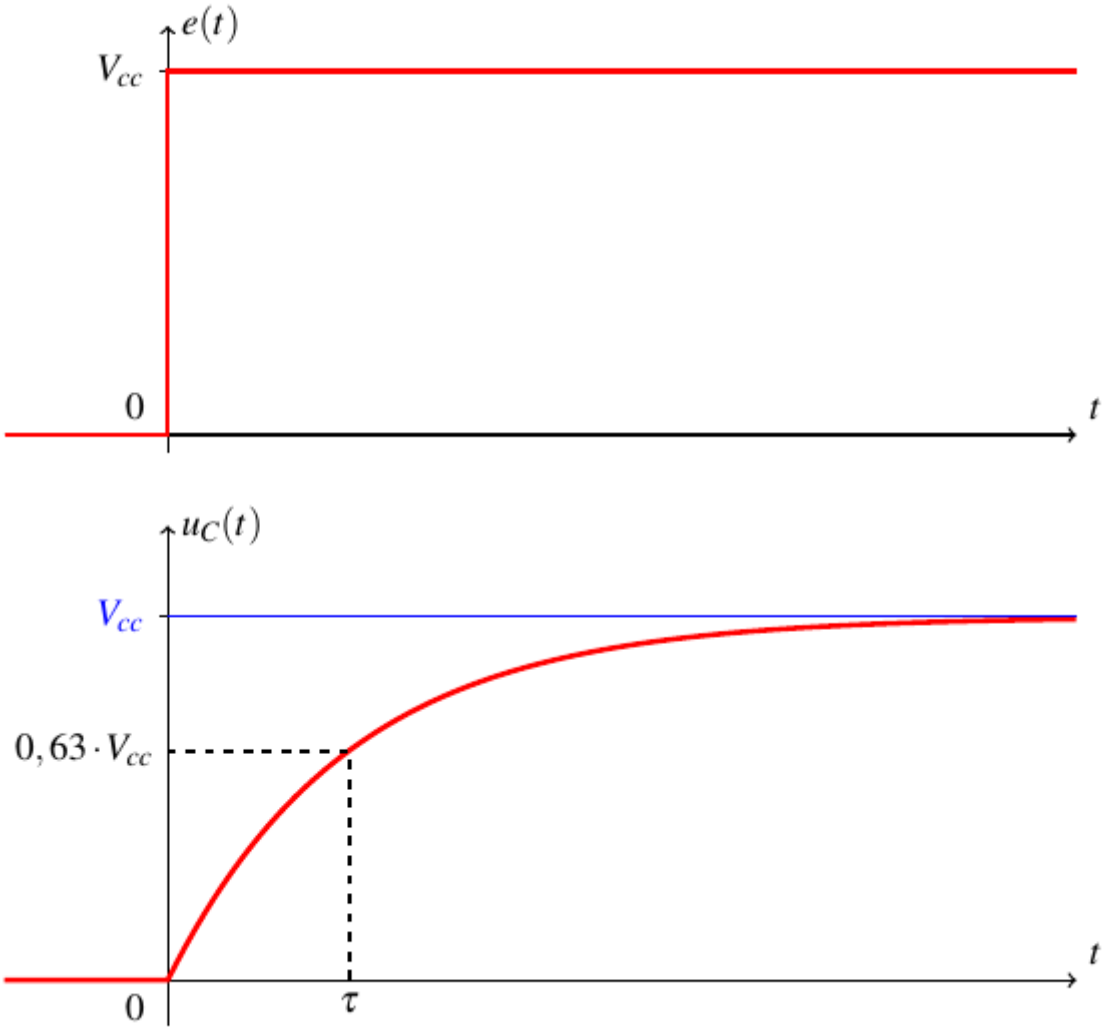



FIG. 32 – Évolution de la tension du condensateur en fonction du temps

Lors de la charge du condensateur C à travers la résistance R sous tension constante $V_{cc}$, le **temps caractéristique** (ou constante de temps) $\tau$ est la durée que prend la tension $u_C$ pour atteindre 63% de sa valeur finale $V_{cc}$.

Avec un microcontrôleur, il est assez facile de mesurer ce temps caractéristique par une **mesure de durée** jusqu'à la

**détection du seuil de 63%** de la valeur finale de la tension du condensateur.

### 6.7.2 Montage

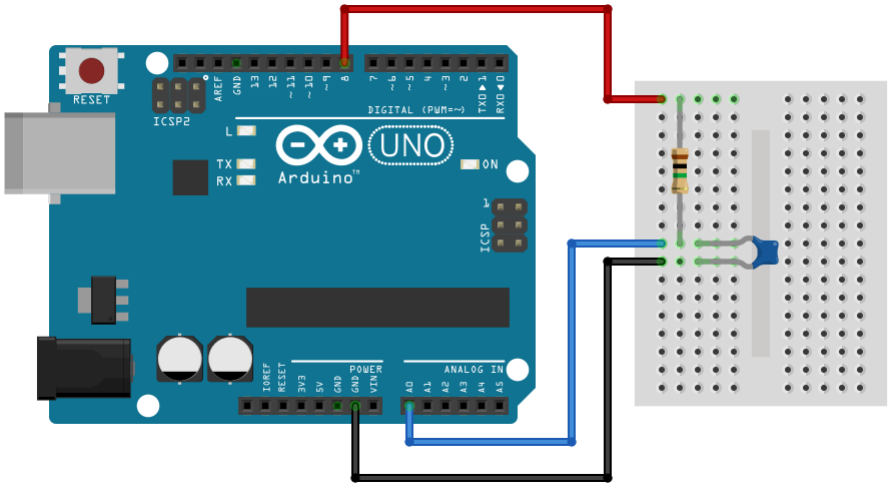



FIG. 33 – Charge d'un condensateur sous tension constante avec un Arduino UNO R3

— La broche digitale 8 charge ou décharge le condensateur à travers la résistance.
— A0 mesure la tension aux bornes du condensateur.

Données :

$$R = 1\,M\Omega \qquad C = 22\,nF$$

### 6.7.3 Programme

```
/*
 *  Mesure de constante de temps d un circuit RC
 */

#define pinE 8

int N = 0;
unsigned long t0;
unsigned long t1;
unsigned long tau;
float C;


void setup() {
  Serial.begin(9600);
  Serial.println("Start");
  pinMode(pinE,OUTPUT);       // Broche digitale en sortie

  digitalWrite(pinE,LOW);     // Décharge condensateur avant mesure
  delay(1000);                // pendant 1 s

  digitalWrite(pinE,HIGH);    // Début charge condensateur
  t0 = micros();              // Mesure instant initial
```

(suite sur la page suivante)

(suite de la page précédente)

```
  while (N<646) {            // Boucle tant que tension inférieure à seuil (0,
↪632*1023=646)
    N=analogRead(A0);        // Lecture tension condensateur
  }

  t1 = micros();             // Mesure instant où seuil atteint
  digitalWrite(pinE,LOW);    // Début décharge condensateur
  tau = t1 - t0;              // Calcul de tau

  Serial.print(tau);          // Début affichage
  Serial.println(" µs");
}

void loop() {
  // Boucle sans fin pas utilisée ici !
}
```

### 6.7.4 A retenir

— La fonction `micros()` renvoie la durée en µs (< 70 min) depuis que la carte Arduino a été mise sous tension. La précision est de 4 µs !
— La boucle `while` (tant que) associée à la fonction `analogRead()` détecte le seuil de la tension du condensateur.

### 6.7.5 Application : mesure d'une capacité

Sachant que le temps caractéristique est défini par la relation :

$$\tau = R \cdot C$$

Le calcul de la capacité C du condensateur est :

$$C = \frac{\tau}{R}$$

If suffit donc d'ajouter cette relation dans le code précédent !

```
/*
 *  Mesure de la capacité d un circuit RC
 */

#define pinE 8

float R = 1000;       // Resistance en kOhm
int N = 0;
unsigned long t0;
unsigned long t1;
unsigned long tau;
float C;


void setup() {
  Serial.begin(9600);
  Serial.println("Start");
  pinMode(pinE,OUTPUT);      // Broche digitale en sortie
```

(suite sur la page suivante)

(suite de la page précédente)

```
  digitalWrite(pinE,LOW);     // Décharge condensateur avant mesure
  delay(1000);                // pendant 1 s

  digitalWrite(pinE,HIGH);    // Début charge condensateur
  t0 = micros();              // Mesure instant initial

  while (N<646) {             // Boucle tant que tension inférieure à seuil (0,
↪632*1023=646)
    N=analogRead(A0);         // Lecture tension condensateur
  }
  t1 = micros();              // Mesure instant où seuil atteint
  digitalWrite(pinE,LOW);     // Début décharge condensateur

  tau = t1 - t0;              // Calcul de tau
  C = tau/R;                  // Calcul de C en nF

  Serial.print(C);            // Début affichage
  Serial.println(" nF");      // Fin affichage
}

void loop() {
  // Boucle sans fin pas utilisée ici !
}
```

# Chapitre 7

# Aller plus loin

## 7.1 Afficheur LCD

Ce sont des afficheurs génériques de 32 caractères disposés sur 2 lignes. Ils ont la particularité d'être simple à mettre en œuvre. Il en existe deux types.

### 7.1.1 Afficheur LCD 16x2 sur port I2C

#### 7.1.1.1 Principe

Ce sont les mêmes afficheurs que précédement mais avec un port série de données (I2C) nécessitant moins de cables (4 en tout).

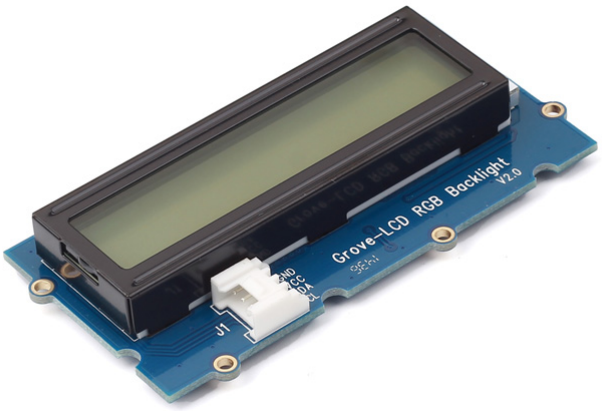



FIG. 1 – Module Grove - LCD RGB Backlight (image : http://wiki.seeedstudio.com)

Chaque afficheur utilise sa propre librairie (ex. `rgb_lcd.h` pour le Grove LCD RGB Backlight) en plus de la librairie `wire.h` qui est obligatoire pour la gestion du port I2C.

#### 7.1.1.2 Montage

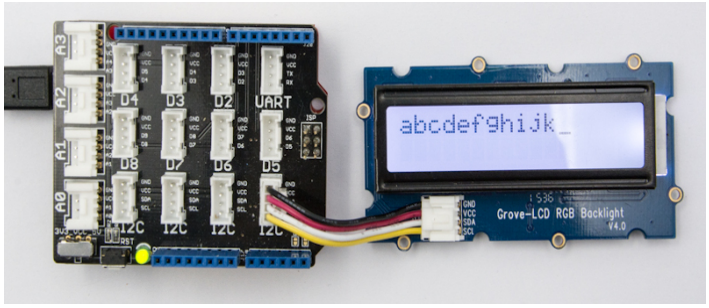



FIG. 2 – Modules Grove (image : http://wiki.seeedstudio.com)

#### 7.1.1.3 Programme en langage Arduino

Télécharger ici le fichier `Grove_LCD_RGB_Backlight-master.zip` pour l'installation de la librairie `rgb_lcd` avant la compilation du programme.

```
/*
 * Exemple affichage sur LCD 2x16 RGB I2C Grove
 */


#include <Wire.h>       // Importation librairie gestion port I2C
#include "rgb_lcd.h"    // Importation librairie gestion afficheur LCD I2C Grove

rgb_lcd lcd;            // Déclaration de l'afficheur

const int colorR = 255; // Couleur fond écran
const int colorG = 0;
const int colorB = 0;

void setup()
{
    lcd.begin(16, 2);                       // Fixe 2 colonnes et 16 caractères/ligne
    lcd.setRGB(colorR, colorG, colorB);     // Fixe couleur de fond
    lcd.print("hello, world!");             // Affiche texte
    delay(1000);                            // Temporisation 1s
}

void loop()
{
    lcd.setCursor(0, 1);                    // Déplace le curseur
    lcd.print(millis()/1000);               // Affiche le temps écoulé en s (timer␣
 ↪interne)
    delay(100);                             // Temporisation 1s
}
```

Pour les versions plus simple sans rétro-éclairage RGB :

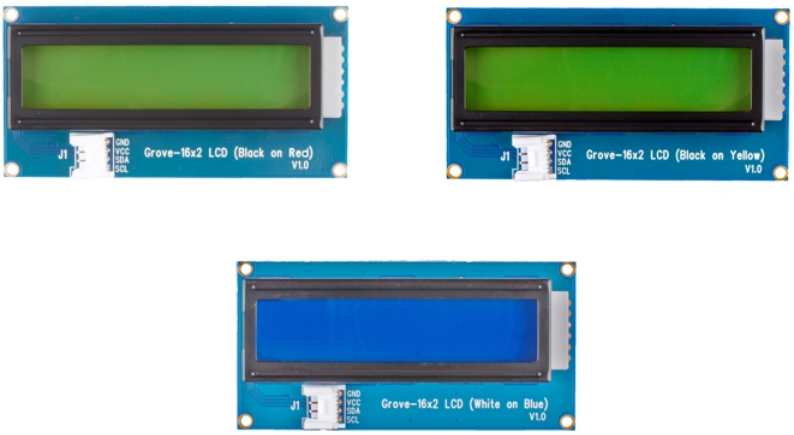



FIG. 3 – Grove LCD 16x12 (image : http://wiki.seeedstudio.com)

```
/*
 * Exemple affichage sur LCD 2x16 I2C Grove sans RGB
 */

#include <Wire.h>
#include "rgb_lcd.h"

rgb_lcd lcd;          // Déclation de l'afficheur LCD branché sur un port I2C

void setup()
{
   lcd.begin(16, 2);  // Initialisation de l'afficheur LCD sur 2 lignes à 16 caractères
}

void loop()
{
   lcd.setCursor(0, 0);         // Placement du curseur
   lcd.print("Bonjour !");      // Affichage de la valeur de la tension
   delay(1000);                 // Pause de 1000 ms
}
```

### 7.1.2 Afficheur LCD 16x2 sur port parallèle

#### 7.1.2.1 Principe

Le pilotage d'un afficheur LCD 16x2 nécessite de 6 broches numériques. Pour éviter un câblage trop complexe, le plus simple est de fixer sur la carte de développement un « shield » afficheur.

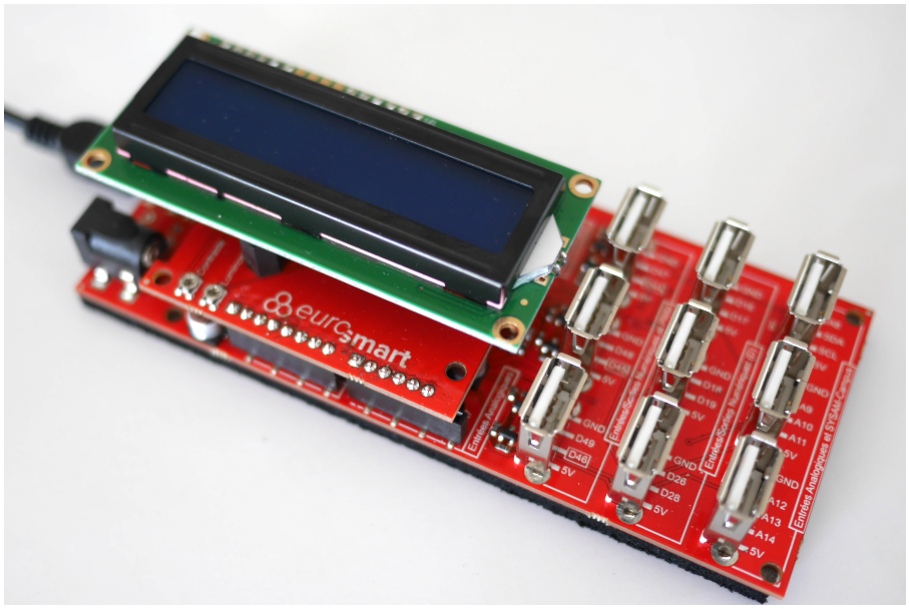



FIG. 4 – Afficheur Educaduino-Lab

#### 7.1.2.2 Programme en langage Arduino (C/C++)

```
/*
 * Exemple d'utilisation d'un écran LCD 16x2 parallèle
 */

#include <LiquidCrystal.h>        // Importation de la librairie LiquidCrystal

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // Brochage de l'afficheur

void setup() {
  lcd.begin(16, 2);               // fixe le nombre de colonnes et de lignes de l␣
 ↪afficheur
}

void loop() {
  lcd.setCursor(5,0);             // place le curseur à la colonne 5 et à la ligne 0
  lcd.print("Bonjour");           // Affiche un texte
  lcd.setCursor(0,1);             // place le curseur à la colonne 0 et à la ligne 1
  lcd.print("tout le monde !");   // Affiche un autre texte
}
```

**Note**

Il sera peut-être necessaire d'installer la librairie `LiquidCrystal` dans le logiciel Arduino.

#### 7.1.2.3 En résumé

| Instruction | Description |
|---|---|
| `#include <LiquidCrystal.h>` | Importe la librairie de gestion de l'afficheur LCD |
| `LiquidCrystal lcd(12, 11, 5, 4, 3, 2)` | Déclare l'afficheur en précisant les numéros de broches |
| `lcd.begin(16, 2)` | Fixe le nombre de colonnes et de lignes de l'afficheur |
| `lcd.setCursor(col,line)` | Positionne le cuseur |
| `lcd.print(variable)` | Affiche le contenu d'une variable à la position du curseur |

Télécharger la `documentation en PDF`.