Centre interuniversitaire de préparation à l'agrégation de Montrouge

MICROCONTROLEURS

2022-2023

Table des matières

Bibliographie :

1 Introduction

1.1 Présentation Arduino

Un Arduino est l'assemblage de deux éléments : un microcontrôleur et un circuit imprimé sur lequel il est connecté et qui permet de l'alimenter, d'utiliser ses entrées/sorties et de l'interfacer via un port USB. Le plan de conception de la plaque est opensource, et c'est un des aspects qui a lancé l'engouement pour l'Arduino il y a quelques années.

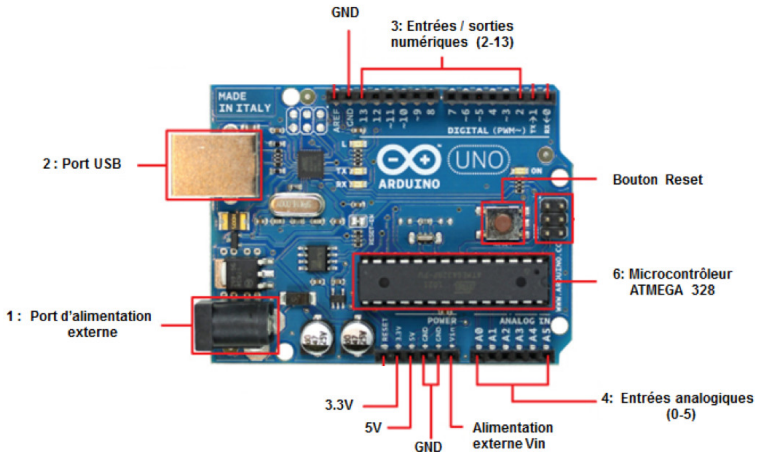


FIGURE 1 – Arduino Uno

On peut décrire rapidement les éléments importants de la photo de l'Arduino Uno :

- On communiquera et on alimentera l'Arduino avec le port USB en le connectant à un ordinateur.
- Le microcontrôleur a 6 entrées analogiques (numérotées de `A0` à `A5`). C'est en général sur ces fiches qu'on branche les capteurs externes.
- Il a également de nombreuses entrées/sorties numériques (numérotées de `2` à `13`), donc qui permettent de recevoir ou d'envoyer des 0/1.
- Pour les circuits électriques, on utilise en général la fiche 5V d'alimentation, et une fiche GND (masse).
- On peut réinitialiser l'Arduino en appuyant sur le bouton Reset.

1.2 Présentation breadboard

Afin de câbler les dipôles et les capteurs au microcontrôleur, on utilise une plaque appelée breadboard (cf. photo). Sur le breadboard, certaines fiches sont reliées les unes aux autres : toutes les fiches d'une ligne sont électriquement liées. Les fiches sur les colonnes externes de la plaque sont également reliées, on les utilise pour l'alimentation si besoin.

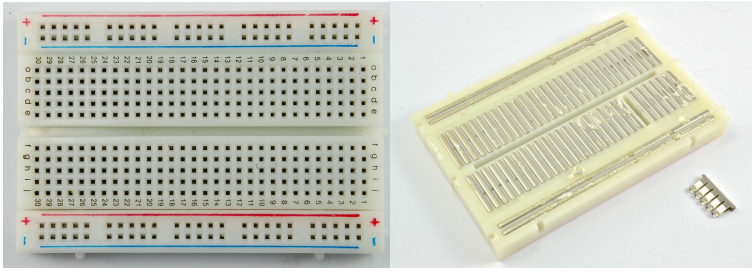


FIGURE 2 – Photographies d'un breadboard (a) face avant (b) face arrière.

1.3 Composants

On utilise différents types de composants avec l'Arduino :

- des dipôles "nus" simples comme des résistances, des condensateurs. Au passage, les résistances étant nues, il faut utiliser le code couleur pour déterminer la valeur de R ;
- d'autres dipôles (diodes, etc.), des tripôles (transistor, etc.) ou des quadripôles (boutons poussoirs, etc.) qu'on utilise également sur le breadboard ;
- des éléments utilisés comme des capteurs, par exemple des photorésistances ou thermistances dont la résistance varie lorsque l'éclairement ou la température change. L'Arduino ne mesure que des tensions, il faut donc souvent un circuit électrique pour convertir la grandeur de mesure (ici la résistance) en tension. On utilise très régulièrement des ponts diviseurs de tension.
- enfin des capteurs fournis avec l'électronique qui permet de lire directement une tension. Ces capteurs utilisent une connectique différente, peut-être plus pratique que les fils simples : la connectique "Grove". Il suffit de relier le capteur au shield de la carte Arduino avec un seul cable, et celui-ci transmet l'alimentation et les entrées/sorties du capteur directement. C'est très pratique. Les entrées "A..." sont des entrées analogiques, et les "D..." des entrées/sorties numériques.

Les capteurs Grove :

- sont issus du Sensor Kit. Toutes les fiches pratiques sont sur ce site.
- Sonde à effet Hall
- Télémètre à ultrasons
- Capteur de température
- Pressiomètre

Les documentations et les spécifications techniques de ces capteurs sont disponibles sur la Dropbox (et bientôt en Notice ?)

1.4 Soft

Afin de programmer les microcontroleurs type Arduino, on utilise le logiciel IDE Arduino qui permet d'écrire le code dans le langage d'Arduino (dérivé de C), de le téléverser sur le microcontrôleur, et de récupérer les informations récupérées via le port Série.

Pour bien prendre en main l'interface, l'onglet "Outils" est important :

- **Port :** permet de spécifier (ou connaître) le nom du port série où est connecté le microcontroleur ;
- **Type de carte :** Permet de préciser le type de carte utilisé (ici uniquement des Arduino Uno)
- **Moniteur série :** Permet d'afficher le rendu du port série. C'est grace à ce moniteur qu'on obtient le retour de l'Arduino.
- **Traceur série :** Idem que le moniteur série sous la forme d'un graphique.

Lorsqu'on écrit un code, on peut tester sa compilation en cliquant sur l'icône *Vérifier*. Puis, lorsqu'on est prêt à le transférer au microcontroleur, on peut cliquer sur *Téléverser*.

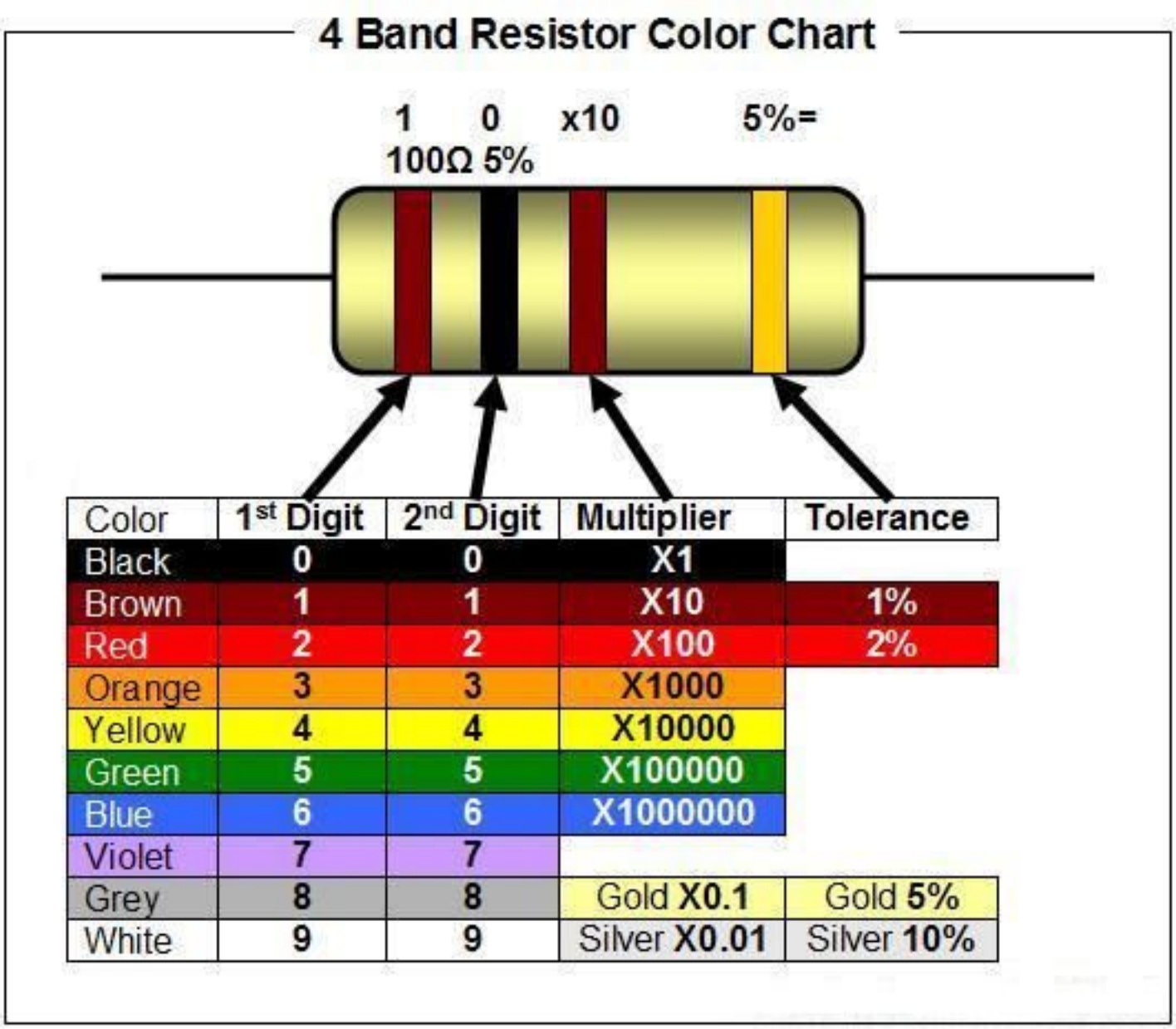
Attention, pour pouvoir afficher les informations envoyées par une carte Arduino il faut que la carte et l'ordinateur se mettent d'accord sur le débit d'information (*baud rate*). Cela se fait du côté Arduino à l'aide de la commande `Serial.begin()` et du côté de l'ordinateur par un menu déroulant dans la fenêtre du moniteur ou du traceur. Si les débits ne correspondent pas, la communication ne pourra pas se faire !

Pour ajouter une librairie, il faut aller dans *Sketch*, *Librairies* et *Ajouter une librairie .ZIP* en sélectionnant la librairie voulue.

1.5 Langage

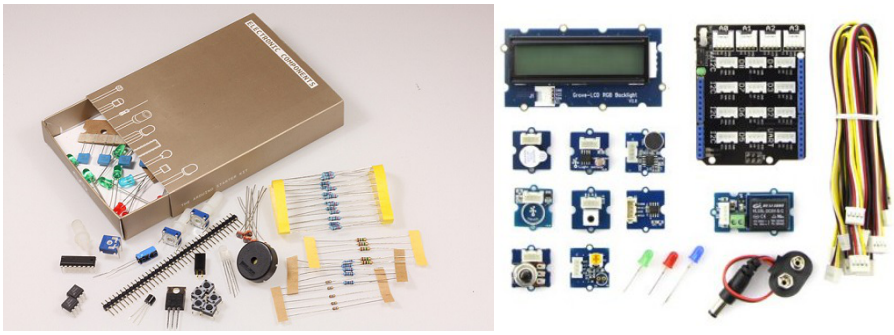
L'objectif n'est pas ici de faire un cours détaillé sur le langage Arduino. On va très pragmatiquement expliquer le fonctionnement d'un code.

Un code Arduino est composé de trois parties essentielles :



Color	1st Digit	2nd Digit	Multiplier	Tolerance
Black	0	0	X1	
Brown	1	1	X10	1%
Red	2	2	X100	2%
Orange	3	3	X1000	
Yellow	4	4	X10000	
Green	5	5	X100000	
Blue	6	6	X1000000	
Violet	7	7		
Grey	8	8	Gold X0.1	Gold 5%
White	9	9	Silver X0.01	Silver 10%

FIGURE 3 – Code couleur des valeurs de résistances.



(a) (b)

FIGURE 4 – (a) des composants nus typiques d'Arduino. (b) des capteurs Grove avec le shield et les câbles adaptés.

- **La définition des variables globales**, qui peuvent être de différents types (dont `int`, `float` et d'éventuelles variations : `long` (grand entier), `double` (grand nombre à virgule flottante), etc.). On peut aussi si besoin y définir des fonctions. Par exemple :

```
// Declaration des variables
unsigned long startTime = 0;          // entier long non signé (positif) dans lequel on stocke
unsigned int  tension = 0;            // entier non signé dans lequel on stockera le signal de
```

- **L'initialisation** contenue dans la fonction `setup()`. On y place les définitions des fiches (les *pin*) utilisées sur la plaque, la vitesse de communication avec le port série... Tous les éléments qui n'ont besoin d'être fait qu'une fois. Par exemple :

```
void setup()
Serial.begin(9600);                  // vitesse de communication avec l'ordinateur
pinMode(8, OUTPUT);                  // le port 8 est défini comme une sortie
pinMode(A0, INPUT);                  // le port A0 est défini comme une entrée
digitalWrite(8, HIGH);               // on fixe le potentiel à la fiche 8 à +5 V
Serial.println("Debut des mesures"); // on communique un petit texte a l'ordinateur
startTime = millis();                // on enregistre le temps du debut de l'experience
```

- *' La boucle*' contenue dans la fonction `loop()` qui contient tout le reste du programme, et qui comme son nom l'indique va se relancer à l'infini tant que l'Arduino est connecté. Cette boucle va contenir toutes les instructions ("lit la valeur du capteur", "change le statut de telle pin", "attends x millisecondes", etc.) :

```
void loop()
tension = analogRead(A0);            // on lit la valeur de la tension en A0
Serial.println(tension);             // on communique cette tension à l'ordinateur
delay(100);                          // on attend 100 ms
```

On utilise très souvent des tests avec `if/else`. On peut lancer une boucle un nombre déterminé de fois avec `for`, ou tant qu'une condition est vérifiée avec `while`. La syntaxe à utiliser est celle du langage C.

Quelques fonctions très utiles :

- `Serial.println(x)` : permet d'afficher la valeur de `x` dans le moniteur série.
- `digitalWrite(pin1, HIGH)` : permet de changer le statut de `pin1`. Les deux modes possibles sont `HIGH` et `LOW`, qui correspondent respectivement à 0 et 5 V.
- `analogRead(pin2)` : permet de lire la valeur de `pin2`, une fiche a priori analogique.
- `millis()` et `micros()` : permettent d'obtenir le temps en millisecondes (resp. microsecondes) depuis l'allumage de l'arduino.
- `delay(t)` et `delayMicroseconds(t)` : permet d'attendre un temps `t` (en millisecondes / resp. en microsecondes), qui peut être un entier ou un nombre fractionnaire.

2 Émission et acquisition de signaux par les ports Entrée/Sortie (I/O)

Dans cette partie on se propose d'utiliser l'Arduino en tant qu'émetteur et récepteur de signaux électroniques (en tension) par le biais de ses ports I/O, puis d'en étudier les limites.

2.1 Limitations et caractéristiques techniques de l'Arduino

Les sources de limitations des microcontrôleurs sont multiples, il faut bien vérifier dans chaque situation laquelle (ou lesquelles) sont dominante(s). On peut en citer quelques unes :

- **Limitations électroniques :** comme tout système électronique, tension de sortie limitée à la tension d'alimentation (5V), courant de sortie limité (40 mA). Attention, l'Arduino n'est pas conçu pour faire de l'électronique de puissance ; veillez à bien respecter les limites constructeur.

- **Limitations numériques :** l'Arduino est une carte numérique et il y a donc des limites de transmission des données liées à la discrétisation (10 bits pour 5V en entrée analogique), la fréquence d'échantillonnage, le temps de communication avec le port sériel USB (baud rate), au temps d'exécution du code, à la mémoire vive (RAM)...

- à cela s'ajoutent des limites techniques diverses : limite de portée (l'Arduino doit être câblé à ses différents capteurs et à son alimentation), limite en température et humidité (on ne peut pas l'utiliser dans tous les environnements)...

Notez que les limites ne sont pas des limites "fondamentales", mais simplement des limites matérielles liées aux compromis faits lors de la conception du microcontrôleur étudié, un Arduino étant principalement conçu pour être simple et surtout peu cher. En général, ses performances sont donc médiocres lorsqu'on les compare à celles d'un appareil dédié.

Microprocesseur	Modèle	ATmega328P
	Fréquence d'horloge	16 MHz
	Résolution compteur (`micros`)	4 µs
	Mémoire vive	2 Kio
	Mémoire flash	32 Kio
Port série	Vitesse de communication	300–115 200 Bd
Ports digitaux (entrée)	Tension d'entrée max	7–12 V
	Impédance d'entrée	~ 100 MΩ
	Temps de lecture	~ 4 µs
Ports digitaux (sortie)	Tension de sortie max	5 V
	Courant de sortie max	20 mA
	Impédance de sortie	~ 50 Ω
	Temps d'écriture	~ 4 µs
Ports analogiques (entrée)	Tension d'entrée max	7–12 V
	Impédance d'entrée	~ 100 MΩ
	Temps d'échantillonage	104 µs
	Résolution	10 bits (5V/1024)

Nota : 1 Kio = 1024 octets, 1 Bd (baud) = 1 symbole/seconde.

FIGURE 5 – Caractéristiques de l'Arduino UNO R3

2.2 Acquisition d'un signal analogique

[1P] Oscilloscope rudimentaire **Attention :** il faut limiter les signaux d'entrée entre 0 et 5V pour ne pas endommager l'Arduino !

Protocole

- Avec un GBF, créer un signal sinusoïdal de basse fréquence (10 Hz) entre 0 et 5V (ne pas se tromper sur le `Vpp` et l'`offset` !), et mettre sur deux colonnes du breadboard le signal.
- A l'aide des bribes de codes du paragraphe d'introduction, écrire un code pour acquérir le signal avec `analogRead` et l'afficher sur le moniteur série de l'Arduino.

[2P] Limite : temps d'échantillonnage et mémoire de l'Arduino L'Arduino à une fréquence d'échantillonnage pour les ports analogiques assez limitée (~ 10 kHz) [1], et si le critère de Shanon (qui dit que la fréquence de la plus grande harmonique du signal doit être inférieure à $f_{\text{éch}}/2$) n'est pas respecté, on peut voir un repliement du spectre.

1. Bien que l'holorge du Atmega328P soit à 16 MHz, les convertisseurs analogique-numérique (CAN) divisent cette fréquence par 2^N, $N = 10$ étant le nombre de bits sur lesquels sont codés le signal. A cela s'ajoute des temps d'exécution dans le microcontrôleur. Le temps

Le temps de communication avec le port série pouvant être limitant, il vaut mieux stocker les données dans une mémoire vive (RAM)[2] qui joue le rôle de tampon (buffer), puis les communiquer après acquisition. Pour cela, utiliser le programme `arduinoscope_buffer`. L'Arduino ayant une faible RAM (2048 octets pour l'Arduino UNO), on ne peut stocker au plus que 1000 entiers (chaque entier pesant 2 octets) : ici avec 250 points de mesure, on utilise 80% de la mémoire vive disponible. Pour comparaison, un oscilloscope Keysight de la collection peut stocker 50 000 points de mesure codés sur 4 octets, soit 200 fois plus de RAM.

Protocole Avec l'Arduinoscope, faire l'acquisition d'un signal triangulaire et observer le repliement du spectre. On pourra voir en log-log la décroissance en $1/f^2$ (pente -2) des composantes harmoniques du triangle ainsi que le repliement avec une pente +2 pour les harmoniques de fréquences plus grandes que $f_{\text{éch}}/2$.

Variante On peut également regarder les limitations liées à la communication série en utilisant `arduinoscope_live` pour différents débits (baudrates), paramétrables avec la commande `Serial.begin(Débit en bauds)`.

2.3 Émission d'un signal numérique

[1P] Générateur de fonctions rudimentaire Protocole

- A l'aide des bribes de codes du paragraphe d'introduction, écrire un code pour émettre un signal créneau avec `analogWrite` mis en mode `LOW` et `HIGH` alternativement et séparé de `delay`.
- Visualiser le signal de sorte à l'oscilloscope.
- Éventuellement utiliser une diode électroluminescente en série avec une résistance ($\sim 2\text{k}\Omega$, attention au sens de la DEL !) pour visualiser le signal de sortie.

Variante possible [AP] : ajouter un potentiomètre pour faire varier le temps de pause manuellement.

[2P] Limite : temps d'écriture des ports digitaux Enlever le délai séparant les créneaux et mesurer le temps d'écriture des ports digitaux (c'est *à peu près* la période des créneaux).

2.4 Application : La charge d'un condensateur

[1P] Programme de base L'intérêt des microcontrôleurs type Arduino réside dans le fait de pouvoir être utilisé avec des dipôles simples d'utilisation courante, et établir des connexions directement sur le breadboard.

On propose une expérience très simple permettant de mesurer la capacité d'un condensateur à l'aide d'un circuit RC.

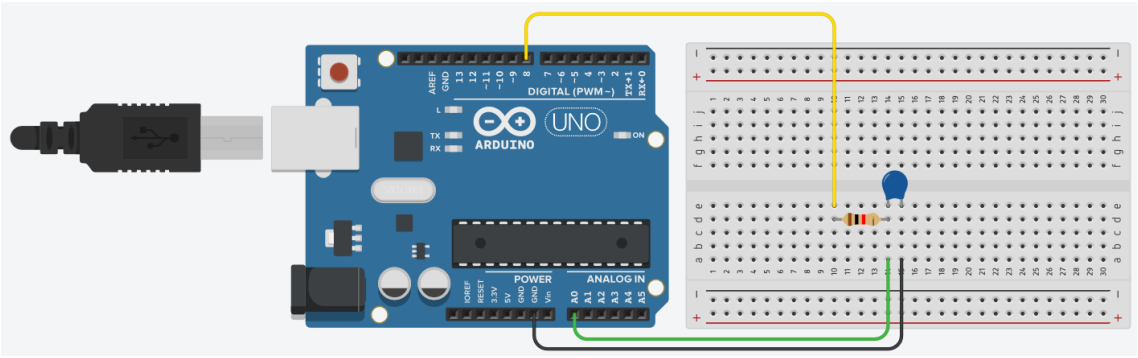


FIGURE 6 – Schéma du montage

Protocole :

d'usine de l'horloge des CAN de l'Atmega328P est légèrement supérieur pour assurer une meilleure régularité : 104 µs soit environ 9600 Hz.

2. La mémoire vive (Random Access Memory en anglais) est une mémoire logique accessible rapidement par le processeur et modifiable en cours d'exécution. Elle est perdue si le processeur n'est plus alimenté en électricité (volatilité).

- Brancher sur le breadboard une résistance et une capacité en série de telle sorte que τ soit de l'ordre de 0,1s.
- Brancher l'entrée du circuit à la PIN 8 de l'Arduino, la sortie à une broche de masse (Ground) et envoyer la tension aux bornes du condensateur à l'entrée "Analog In" A0.

Programme : Le programme condensateur_Montrouge réalise les étapes suivantes :

- Initialisation : on définit la pin 8 une pin active de sortie, on connecte le microcontrôleur au port série, puis on éteint la pin 8 si jamais elle était allumée. On attend 1000ms. Puis on impose une tension sur le système via la pin 8.
- Boucle : on a défini un pas de quantification (la variable "intervale"). Après ce Δt, on regarde si le condensateur est en charge (état 1), décharge (état 2), ou si l'expérience est finie (état 3). Pendant la charge, on affiche la tension sur le moniteur série, jusqu'à ce que la tension atteigne 63% de la tension maximale (à savoir 1023). On mesure le temps écoulé, on obtient directement τ. Puis on décharge le condensateur en basculant dans l'état 2. Et on recommence l'expérience en basculant dans l'état 1 sauf si on a fait suffisamment de fois l'expérience.

Valeurs : On peut utiliser un condensateur de 100nF avec une résistance de 1MOhm par exemple, on obtient un temps caractéristique de 0,1s. La répétition de l'expérience montre que la valeur obtenue pour le temps de montée est stable. Choisir des composants tels que τ est plus petit peut mettre en avant l'intérêt de répéter l'expérience, et de faire une moyenne des valeurs obtenues (incertitudes statistiques de type A).

[2P] Limite : impédances d'entrée et de sortie des ports I/O Les ports entrées/sorties du microcontrôleur possèdent des impédances caractéristiques fixées par le constructeur, principalement afin d'éviter la circulation de courants trop importants. Suivant si le port est en mode entrée (`INPUT`) ou sortie (`OUTPUT`), les ports sont mis sur des états de haute ou de basse impédance dont les circuits équivalents sont les suivants :

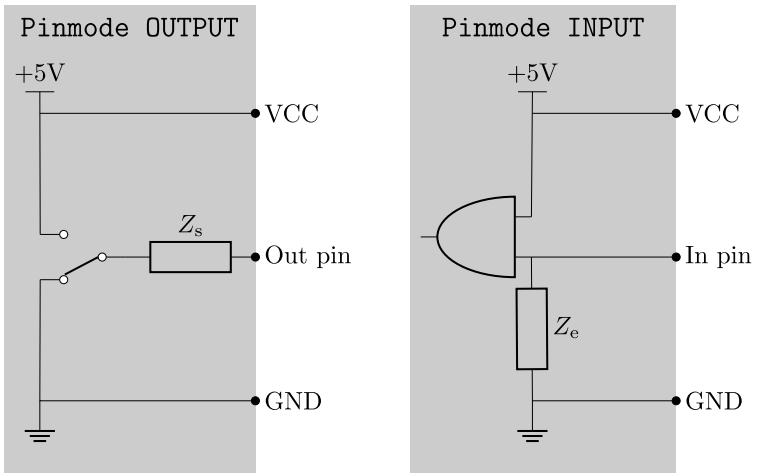


FIGURE 7 – Circuit équivalent des ports I/O du microcontrôleur suivant le mode, mettant en évidence les impédances d'entrée Z_e et de sortie Z_s. Dans la configuration `OUTPUT`, l'interrupteur commute entre 0 et +5V selon que la sortie soit en 0 (`LOW`) ou 1 (`HIGH`). Dans la configuration `INPUT`, le tripole arrondi représente le système de conversion analogique-numérique du microcontrôleur.

Dans la situation précédente, prenant en compte des impédances d'entrée et de sortie, le circuit équivalent est le suivant :

On a $Z_s \simeq 50\ \Omega$ et $Z_e \simeq 100\ \text{M}\Omega$ (voir les caractéristiques techniques données précédemment).

Protocole

- Ajuster les valeurs de R et C de la manipulation précédente afin d'obtenir Z_s et Z_e.

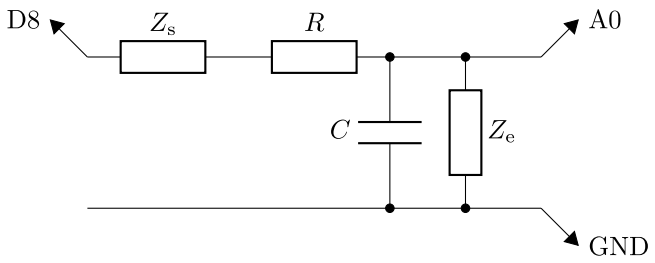


FIGURE 8 – Modélisation des impédances d'entrée et de sortie

2.5 [AP] Utilisation d'un pont diviseur : le cas d'une thermistance

Il est fréquent que le transducteur utilisé convertisse la grandeur d'intérêt en autre chose qu'une tension, par exemple une variation de sa résistance (thermistance, Pt100, photo-résistance, etc.) Dans ce cas assez courant, on cherche à "transformer" une résistance en tension, et on utilise un pont diviseur de tension. L'idée est de mettre en série une résistance R connue et le dipôle dont la résistance R_{dip} varie autour de la valeur de R. En appliquant une tension U_{in} à l'ensemble (5V par exemple), on mesure la tension aux bornes du dipôle qui vaut

$$U = \frac{R_{\text{dip}}}{R + R_{\text{dip}}} U_{\text{in}} \tag{1}$$

On obtient ainsi la valeur de la résistance

$$R_{\text{dip}} = \frac{U}{U_{\text{in}} - U} R \tag{2}$$

Après il faut connaître le lien entre la variation de la résistance et la grandeur mesurée. Pour les thermistances fournies avec le setup Arduino, on pourra aller voir la fiche technique (pénible). Sinon on peut utiliser des thermistances de la collection.

3 Utilisation de capteurs intégrés

On dispose de plusieurs capteurs, faciles à utiliser à l'aide de la connective Grove qui est "plug and play" : on branche délicatement le shield sur l'Arduino, on connecte les capteurs, on ouvre le programme adapté à chaque capteur dans l'IDE Arduino et le moniteur série donne accès aux valeurs du capteur.

Chaque fiche des connecteurs est relié à une fiche de l'Arduino originale (la correspondance est indiquée sur le site du constructeur). Les connecteurs Grove ne sont qu'une manière pratique de brancher les composants à la carte, ce qui permet d'éviter d'avoir une "jungle de fils". Le shield comporte en outre un interrupteur au-dessus du port de connexion au PC qui permet de changer la tension d'alimentation des capteurs, entre 5 V et 3.3 V. **On prendra toujours garde à alimenter les capteurs avec la tension convenable**, sous peine de les détériorer et de fausser les mesures.

Les différents capteurs que l'on peut connecter à la carte communiquent avec elle via plusieurs types d'interface

- Les capteurs à interface analogiques (comme le capteur de pression) sont les plus simples : Une fois alimentés, ils imposent une tension sur une fiche analogique qu'il suffit de lire pour faire la mesure.
- Les capteurs à interface logique (comme le télémètre à ultrasons) : Il faut les alimenter et leur envoyer un signal de commande, puis sur la même fiche attendre leur signal de réponse.
- Les capteurs à interface I2C (comme l'accéléromètre) sont les plus sophistiqués : Il faut les alimenter puis échanger des informations via un protocole complexe impliquant deux fiches [3]. Pour les utiliser, il est nécessaire d'utiliser la bibliothèque `Wire` d'Arduino.

3. La fiche SDA (Serial Data Line), qui sert à l'échange de données, et la fiche SCL (Serial Clock Line) pour la synchronisation des horloges.

3.1 Capteur de pression

On utilise le capteur de pression (description et caractéristiques techniques sur le site Gotronic).

Après avoir placé délicatement le shield sur la carte Arduino, connecter le capteur sur le port A0 du shield. Téléverser le programme `pression_montrouge` sur le microcontrôleur, il permet d'afficher la valeur de la pression dans le moniteur série. Le capteur nécessite un étalonnage : pour cela, il suffit de modifier la variable `offset` représentant le décalage de tension du capteur. Par exemple, on pourra s'assurer que la tension affichée lorsque le capteur n'est pas connecté à la seringue est bien la pression atmosphérique.

Fonctionnement du capteur Le capteur de pression utilisé comporte un élément piézorésistif, c'est à dire dont la résistance change avec la pression appliquée. La mesure de pression se fait donc via une mesure de résistance (souvent à l'aide d'un pont de Wheatstone), dont la valeur est donnée sous la forme d'une tension entre la broche SIG et la broche GND du capteur.

[AP] Sur ce dispositif particulièrement simple, il est possible d'"émuler" l'Arduino en imposant une tension de 5 V entre les fiche VCC et GND, et en observant à l'aide d'un oscilloscope la tension sur la pin SIG.

[1P] Vérification de la loi de Boyle-Mariotte Afin de vérifier que pour un système fermé à température constante, pV = cste, on connecte le capteur à une seringue dont le piston est placé **à mi-course**. Cela permettra d'augmenter ou diminuer la pression tout en restant dans une gamme raisonnable de pressions . Mesurer la pression pour différentes positions du piston. Lors de l'ajustement de la courbe P=f(V), attention à considérer l'intégralité du volume du gaz (en particulier celui dans le tube !)

[2P] Limite : quantification et résolution du convertisseur analogique-numérique La quantification sur 1024 bits du signal devient manifeste en utilisant le grapheur série. Pour y remédier, on peut effectuer une moyenne sur plusieurs valeurs (essayer par exemple au lieu de faire une mesure d'en faire 10 à la suite (en utilisant une boucle `for`)) et de renvoyer la moyenne.), au prix d'une diminution la fréquence d'acquisition maximale.

3.2 Télémètre ultrasonore

On dispose d'un télémètre à ultrasons. Celui-ci utilise un émetteur et un récepteur ultrasons pour mesurer sa distance à un objet. Après avoir placé délicatement le shield sur la carte Arduino, connecter le capteur sur le port D4 du shield. Téléverser le programme `ultrasons_montrouge` sur le microcontrôleur, il permet d'afficher la distance entre le capteur et la surface la plus proche dans le moniteur série. Pour que le lien entre le temps et la distance soit bon, modifier la variable `c` qui représente la vitesse du son dans l'air, en m/s.

Fonctionnement du capteur Le capteur comporte un émetteur et un récepteur ultrasonique, qui échangent un *burst* de 8 oscillations à 40 kHz. L'émission et la détection des ultrasons sont faits par un petit microcontrôleur (appelé MCU, pour MicroController Unit) interne au capteur.

[1P] Étude du capteur En utilisant un récepteur ultrasonique et un oscilloscope de la réserve et en cachant l'un puis l'autre des piézos, identifier l'émetteur et le récepteur.

Réaliser plusieurs mesures de distance en éloignant de plus en plus le télémètre d'une paroi et tracer la distance mesurée en fonction de la distance réelle. Que dire de la précision de la mesure ? Identifier et quantifier les sources d'erreur systématiques et statistiques.

[2P] Écoulement de Torricelli Mettre le capteur tête vers le bas en haut d'un récipient rempli d'eau percé d'un petit trou en bas (par ex. une des éprouvettes munies d'un robinet de la collection). Le télémètre peut mesurer la hauteur d'eau en fonction du temps, on peut récupérer les données et ajuster pour retrouver la loi de Torricelli : $\sqrt{h}$ est une fonction affine du temps, avec h la hauteur de l'interface par rapport au trou.

Référence complémentaire

- voir Fruchart, Lidon *et al*, *Physique expérimentale*, p. 422

3.3 Accéléromètre

On dispose d'un accéléromètre (LIS3DHTR), c'est le capteur le plus sophistiqué de la boîte Arduino. Après avoir placé délicatement le shield sur la carte Arduino et choisi une **tension d'alimentation de 3.3 V**, connecter le capteur sur un des ports I2C du shield. Téléverser le programme `acceleration_ressort_montrouge` sur le microcontrôleur, il permet d'afficher l'accélération selon les trois axes sur le grapheur série. Les valeurs sont indiquées en pourcentages de g, l'accélération de la pesanteur : une mesure de 100 indique une accélération de 9.81 m/s².

Au début du programme (dans la fonction `setup`), il faut spécifier la plage d'acquisition à l'aide de la fonction `LIS.setFullScaleRange()` et la fréquence d'acquisition avec `LIS.setOutputDataRate()`.

Fonctionnement du capteur Le capteur en lui même est un élément très complexe qui comporte un microcontrôleur autonome, des oscillateurs lui permettant d'avoir une fréquence d'acquisition propre, un système de mesure qui lui permet d'avoir une précision sur 16 bits, des intégrateurs pour donner la vitesse et la position, un capteur de température qui lui permet de fonctionner de manière asservie en température entre -40 et 85 degrés, etc...

La partie "accéléromètre" à proprement parler est réalisée à l'aide de MEMS (MicroElectroMechanical Systems) capacitifs. Pour chaque direction, une masse test (ou masse séismique) de quelques micromètres est reliée à des ressorts fortement amortis. Elle est placée de manière à équilibrer des demi-ponts de capacités, alimentés symétriquement par des signaux oscillant à la fréquence d'échantillonnage. Lorsqu'une force s'applique sur la masse cette dernière se déplace en proportion, ce qui déséquilibre les demi-ponts de capacités. La tension en milieu de pont est alors proportionnelle au déplacement relatif de la masse séismique (à vérifier !), elle est ensuite amplifiée et mesurée.

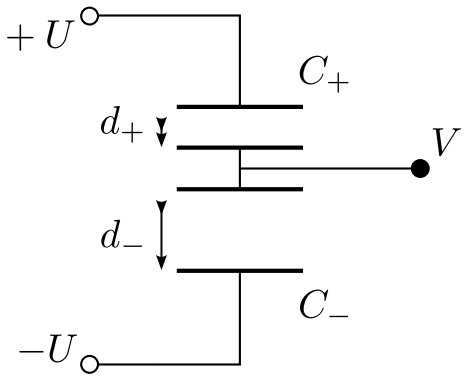


FIGURE 9 –

L'accéléromètre est donc en fait un dynamomètre, qui mesure *toutes* les forces s'exerçant sur la masse séismique : les forcesssssssssssssssssssssssssssssssssssssssss inertielles dûes à l'accélération *mais aussi la gravité* (ces forces étant d'après le principe d'équivalence indifférentiables).

[1P] Système masse-ressort Fixer l'accéléromètre à une masse accrochée verticalement à un support par un ressort, en faisant attention à ce que les broches métalliques de la face intérieure ne soient pas en contact avec du métal, ce qui peut créer des courts-circuits. Mesurer l'accélération selon l'axe vertical en mettant la variable `mode_enregistrement` à `true` et en choisissant le temps d'acquisition avec la variable `T_enregistrement`. On pourra utiliser le programme `acceleration_ressort_moyennage_montrouge` qui est plus précis. Enregistrer les valeurs et les afficher avec Qtiplot.

En déduire la raideur du ressort. Comparer à la valeur tabulée, ou mesurée avec une autre méthode. En quoi l'ajout de l'accéléromètre perturbe-t-il la mesure ? Quels sont les avantages de ce système par rapport à d'autres méthodes de mesure du mouvement de la masse, par exemple une prise de vue optique à la caméra avec tracking ?

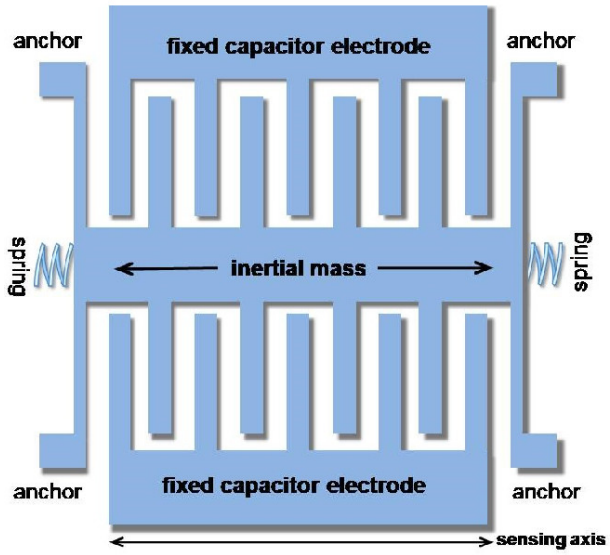


FIGURE 10 – Schéma de principe d'un accéléromètre capacitif.

4 Aller plus loin ? L'Arduino autonome

Un intérêt important aux microcontrôleurs est qu'ils peuvent être utilisés en autonomie :

- soit en tant que capteur autonome, dont on récupérera les données ultérieurement ;
- soit en tant que "mini-ordinateur", le microcontrôleur peut agir sur des actionneurs en fonction des retours donnés par ses capteurs, et ainsi faire des rétroactions automatiques. Par exemple en gardant la température d'une enceinte, une intensité lumineuse ou une vitesse de rotation constante, ou en réagissant lorsque qu'une grandeur dépasse une valeurs seuil pré-déterminée.

Un exemple d'utilisation autonome : alimenter l'Arduino avec des piles, connecter le capteur de pression atmosphérique et un écran à cristaux liquides. En plaçant l'ensemble dans la cloche à vide, on peut mesurer la pression en fonction du temps, voire l'enregistrer.

Rq : il n'y a pas (pour l'instant) de programme correspondant à cette expérience. N'hésitez pas à vous manifester si vous avez des suggestions.

5 (Appendice) Codes Arduino

5.1 Mesure du temps moyen de charge d'un condensateur

Ce programme permet de mesurer le temps que met un condensateur à se charger. Le système doit être branché comme suit.

```
(pin 8) --- [ R ] ------- (pin A0)
                    |
                    = C
                    |
                (pin GND)
```

La pin 8 fournit la tension d'alimentation (0 ou 5 V).

La pin A0 sert à la mesure de la tension.

Le programme effectue plusieurs cycles de charges, et mesure le temps que met la tension aux bornes du condensateur pour atteindre une valeur seuil.

Attention, ce programme fonctionne à 115200 baud, à régler sur l'affichage moniteur (Crtl+Shift+M).

```
float const pauseEntreLesMesures = 0.; // Nombre de millisecondes entre chaque mesure de la ten
int const pauseEntreCycles = 500; // Nombre de millisecondes entre chaque cycle de mesure.

float const seuil = 0.95; // Seuil de charge/decharge a partir duquel on considere la charge te

int const nombreDeCyclesDeMesure = 10; // Nombre de cycles de mesure à faire pour moyenner.

int tension; // Variable qui stocke la tension sous forme d'un entier codé sur 10 bits (0 V = 0

unsigned long t0; // Variable qui stocke le temps au debt d'une sequence de mesure.
float tempsDeCharge[nombreDeCyclesDeMesure]; // Les temps de charge pour chaque cycle

void setup() {
  pinMode(8, OUTPUT); // alimentation du condensateur
  Serial.begin(115200);

  // DECHARGE INITIALE DU CONDENSATEUR
  digitalWrite(8,LOW); // La tension aux bornes du condensateur est mise a 0

  tension = 1023;
  while (tension > 0.5 * 1023 ) {
    tension = analogRead(A0);
    delay(10);
  }

}

void loop() {

  // *** MESURE DES TEMPS DE CHARGE / DECHARGE ***

  for (int i_mesure = 0 ; i_mesure < nombreDeCyclesDeMesure ; i_mesure += 1) {

    delay(pauseEntreCycles);

    Serial.print("Cycle ");
    Serial.print(i_mesure+1);
    Serial.println(" :");

    // CHARGE
    digitalWrite(8,HIGH); // La tension aux bornes du condensateur est mise a 5 V
    t0 = micros();

    while (1) {
      tension = analogRead(A0);
      if (tension > seuil*1023) {
        tempsDeCharge[i_mesure] = (float)((micros() - t0))/1000;
        break;
      }
      delay(pauseEntreLesMesures);
    }
```

```
    Serial.print("Temps de charge : ");
    Serial.print(tempsDeCharge[i_mesure],3);
    Serial.println(" ms");

    // DECHARGE
    digitalWrite(8,LOW); // La tension aux bornes du condensateur est mise a 0

  tension = 1023;
  while (tension > 0.5 * 1023 ) {
    tension = analogRead(A0);
    delay(10);
  }

  }

  Serial.println();

  // *** AFFICHAGE DES RESULTATS ***

  Serial.print("Sur ");
  Serial.print(nombreDeCyclesDeMesure);
  Serial.println(" cycles :");

  // Calcul temps moyen de charge
  float tempsDeChargeMoyen = 0;
  for (int i=0 ; i < nombreDeCyclesDeMesure ; i += 1) {
    tempsDeChargeMoyen += tempsDeCharge[i];
  }
  tempsDeChargeMoyen = tempsDeChargeMoyen / nombreDeCyclesDeMesure;
  Serial.print("Temps de charge moyen : ");
  Serial.print(tempsDeChargeMoyen,3);
  Serial.println(" ms");

  // Fin du programme, on ne fait plus rien
  while (1) {
    delay(10000);
  }
}
```

5.2 Capteur de pression

Ce programme permet de mesurer la pression, en mesurant la tension sur le pin `A0`.

Pour afficher un graphe continu : choisir `intervalleEntreMesures` court (de l'ordre de 10 (10 ms)) et afficher le grapheur (Crtl+Shift+L).

Pour afficher des valeurs de pression : choisir `intervalleEntreMesures` long (de l'ordre de 1000 (1 s)) et afficher le moniteur (Crtl+Shift+M).

Ce capteur fonctionne en 5V, bien penser à mettre l'interrupteur du shield Grove sur la bonne position !

```
float const intervalleEntreMesures = 10; // Intervalle entre les mesures, en ms.

int const pin = A0; // Identifiant de la 4-broche du shield Grove
```

```
double const sensibilite = 6.4; // Sensibilite du capteur : 6.4 mV / kPa
double const offset = 152; // Offset du capteur : À étalonner, normalement entre 88 et 313 mV

void setup() {
  Serial.begin(115200); // Ce programme fonctionne à 115200 baud, à régler sur l'affichage moni
}

void loop() {
  // On lit la valeur de la tension (entier sur 1024 values) sur la pin A0.
  int rawValue = analogRead(pin);

  // On obtient la tension delivree (en mV)
  double rawVoltage;
  rawVoltage = (double) rawValue * 5000 / 1023;

  // On transforme la tension en pression
  double pression;
  pression = (rawVoltage - offset) / sensibilite;

  Serial.print("Pression (kPa) "); Serial.println(pression);
  delay(intervalleEntreMesures);
}
```

5.3 Télémètre ultrasonore

Ce programme permet de mesurer la distance du télémètre a la surface réfléchissante (pour les ondes sonores) la plus proche de lui. Le télémètre fonctionne en envoyant un train d'onde consistant en 8 oscillations a 40 kHz via le piézo émetteur puis en écoutant avec le piézo récepteur pour pouvoir mesurer le temps de trajet du son.

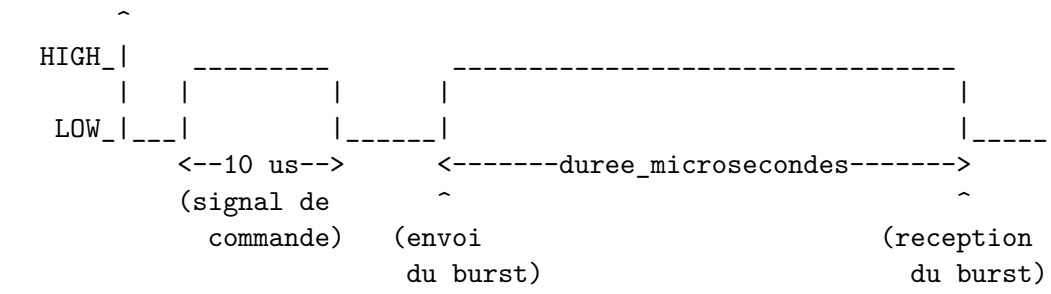
Le télémètre doit être branché sur la sortie `D4` du shield Grove.

La communication se fait a l'aide d'une pin, celle de signal (stockée dans la variable `pinSig`).

Le signal de commande est un creneau TTL de 5 V (`HIGH`) durant 10 us.

En réponse, le télémètre renvoie 0 V (`LOW`) pendant sa préparation, puis impose 5 V (`HIGH`) sur la pin de signal tant que le pulse est en l'air et retombe à 0 V (`LOW`) quand il revient.

Exemple de la tension sur `pinSig` pendant une séquence de mesure :



Attention, ce programme fonctionne à 115200 baud, à régler sur l'affichage moniteur ou plotteur.

```
float const pauseEntreLesMesures = 1000; // Nombre de millisecondes entre chaque mesure de la d

int const pinSig = 4;  // Brancher le capteur ultrason sur l'épingle D4

double const c = 343.0; // Vitesse du son (m/s)
```

```
void setup()
{
Serial.begin(115200);
}

void loop()
{

  long duree_microsecondes;
  duree_microsecondes = mesurerUnTempsDeTrajet();

  double duree_millisecondes;
  duree_millisecondes = ( (double) duree_microsecondes ) / 1000;

  Serial.print("Temps d'aller_retour : ");
  Serial.print(duree_millisecondes, 3);
  Serial.println(" ms");


  double distance_mm;
  distance_mm = duree_millisecondes * c / 2;

  Serial.print("Distance : ");
  Serial.print(distance_mm, 2);
  Serial.println(" mm");

  Serial.println();

  delay(pauseEntreLesMesures);
}


/*
 * Cette fonction fait une mesure de temps de trajet aller-retour d'un burst d'ultrasons à 40 k
 *
 * Elle renvoie le temps écoulé entre l'envoi du burst et sa reception, en microsecondes.
 */
long mesurerUnTempsDeTrajet() {

    // Tout d'abord, il faut envoyer un signal au telemetre pour lui dire de lancer le protocol
    // On commence a mettre le pin en mode OUTPUT et on impose une tension nulle.
    pinMode(pinSig, OUTPUT);
    digitalWrite(pinSig, LOW);
    delayMicroseconds(5);
    // Le signal d'envoi est un creneau de 5 V pendant 10 us.
    digitalWrite(pinSig, HIGH);
    delayMicroseconds(10);
    digitalWrite(pinSig, LOW);
    // Ensuitew on ecoute la reponse du telemetre
    pinMode(pinSig, INPUT);
```

```
    long timeout = 1000000L; // Le delai maximum sans qu'il ne se passe rien est 1 s. Si ce tem

    long debut_sequence = micros();

    // wait for any previous pulse to end
    while (digitalRead(pinSig) == HIGH) {
      if ((micros() - debut_sequence) >= timeout) { return 0; } // Si le timeout est depassé, r
    }

    // wait for the pulse to start
    while (digitalRead(pinSig) == LOW) {
      if ((micros() - debut_sequence) >= timeout) { return 0; } // Si le timeout est depassé, r
    }

    long pulseBegin = micros(); // Date d'envoi du burst

    // wait for the pulse to stop
    while (digitalRead(pinSig) == HIGH) {
      if ((micros() - debut_sequence) >= timeout) { return 0; } // Si le timeout est depassé, r
    }

    long pulseEnd = micros(); // Date de reception du burst

    return pulseEnd - pulseBegin;
}
```

5.4 Accéléromètre LIS3DHTR

”’ ATTENTION, CE CAPTEUR FONCTIONNE SUR 3.3 V. RÉGLER EN CONSÉQUENCE L’INTERRUPTEUR SUR LE SHIELD GROVE. UNE TENSION PLUS ÉLEVÉE DÉGRADE LE CAPTEUR ET ENTRAÎNE DES MESURES FAUSSES ”’

Attention, ce programme fonctionne à 115200 baud, à régler sur l’affichage moniteur ou plotteur.

Ce programme renvoie les accélérations lues par le capteur.

Brancher ce capteur sur une sortie I2C.

Attention, ce programme necessite l’installation de la librairie LIS3DHTR. Pour l’installer : Sketch -> Include Library -> Add .ZIP Library -> sélectionner le fichier "Seeed_Arduino_LIS3DHTR-master.zip" dans le dossier "bibliotheques"

Utiliser Crtl+Maj+L pour afficher le graphe. Utiliser Crtl+Maj+M pour afficher le moniteur.

MODE VISUALISATION

- Mettre la variable mode_enregistrement à false
- Mettre la variable show_x (resp. y, z) à true/false pour afficher/cacher l’accélération selon x (resp. y, z)
- Téléverser le programme
- Afficher le graphe (Crtl + Maj + L)
- Vérifier que le débit est correct (115200 baud)

MODE ENREGISTREMENT Pour utiliser le mode enregistrement :

- Mettre la variable mode_enregistrement à true
- Choisir la valeur de T_enregistrement, le temps d’enregistrement en ms

- Téléverser le programme
- Afficher le moniteur (Crtl + Maj + M)
- Vérifier que le débit est correct (115200 baud)

Pour lancer un enregistrement - *' Attention, ceci ne marche qu'avec les versions de l'IDE Arduino < 2.0!! Sans cela, la commande Crtl + A ne fonctionne pas...*'

- Cliquer sur 'Effacer la sortie' puis appuyer sur le bouton 'reset'.
- Attendre le temps d'enregistrement
- Sélectionner tout (Crtl + A) et copier (Crtl + C)
- Coller dans un fichier texte
- Enjoy

Le temps est affiché en ms, les accélérations sont données en $g / 100$.

```
// This example shows the 3 axis acceleration.
#include "LIS3DHTR.h" // La bibliotheque du capteur
#include <Wire.h>
LIS3DHTR<TwoWire> LIS; // Protocole IIC (Inter_Integrated Circuits)
#define WIRE Wire

// Le délimiteur pour les données enregistrées
//#define SPACER " ; "
#define SPACER "\t"

bool mode_enregistrement = true; // Mettre true pour enregistrer, false pour juste afficher
float T_enregistrement = 10000; // La duree d'enregistrement (en ms)

bool show_x = true;
bool show_y = true;
bool show_z = true;

float t = 0; // Le temps en ms
float acc_x = 0; // L' acceleration selon x
float acc_y = 0; // L' acceleration selon y
float acc_z = 0; // L' acceleration selon z

void setup() {
    Serial.begin(115200); // Travail a 115200 baud
    while (!Serial) {}; // Attendre l'initialisation de la liaison avec l'ordi
    LIS.begin(WIRE, 0x19); //Initier la liaison IIC (entre le capteur et la carte)
    delay(100);
      LIS.setFullScaleRange(LIS3DHTR_RANGE_2G); // le plus petit range pour plus de precision
    //  LIS.setFullScaleRange(LIS3DHTR_RANGE_4G);
    //  LIS.setFullScaleRange(LIS3DHTR_RANGE_8G);
    //  LIS.setFullScaleRange(LIS3DHTR_RANGE_16G);

    //  LIS.setOutputDataRate(LIS3DHTR_DATARATE_1HZ);
    //  LIS.setOutputDataRate(LIS3DHTR_DATARATE_10HZ);
    //  LIS.setOutputDataRate(LIS3DHTR_DATARATE_25HZ);
    // LIS.setOutputDataRate(LIS3DHTR_DATARATE_50HZ);
//      LIS.setOutputDataRate(LIS3DHTR_DATARATE_100HZ);
      LIS.setOutputDataRate(LIS3DHTR_DATARATE_200HZ); // Frequence d'aquisition adaptee a la fr
```

```
//      LIS.setOutputDataRate(LIS3DHTR_DATARATE_1_6KHZ);
//      LIS.setOutputDataRate(LIS3DHTR_DATARATE_5KHZ);
    LIS.setHighSolution(true); //High solution enable

    if (mode_enregistrement) {
      Serial.print("t"); Serial.print(SPACER);
      Serial.print("acc_x"); Serial.print(SPACER);
      Serial.print("acc_y"); Serial.print(SPACER);
      Serial.print("acc_z"); Serial.print(SPACER);
      Serial.println();
    }
}

void loop() {
    if (!LIS) {
        Serial.println("LIS3DHTR didn't connect.");
        while (1);
        return;
    }

    //Chopper les données
    acc_x = LIS.getAccelerationX() * 100;
    acc_y = LIS.getAccelerationY() * 100;
    acc_z = LIS.getAccelerationZ() * 100;

    // Afficher les données
    if (mode_enregistrement) {
      Serial.print( millis() ); Serial.print(SPACER);
      Serial.print( acc_x ); Serial.print(SPACER);
      Serial.print( acc_y ); Serial.print(SPACER);
      Serial.print( acc_z ); Serial.print(SPACER);
      Serial.println();
    } else {
      if (show_x) {Serial.print("x:"); Serial.print(   acc_x   ); Serial.print(" ");}
      if (show_y) {Serial.print("y:"); Serial.print(   acc_y   ); Serial.print(" ");}
      if (show_z) {Serial.print("z:"); Serial.print(   acc_z   ); Serial.print(" ");}
      Serial.println(); // Passer a la ligne
    }

    t += 10;
    delay(10);

    if (mode_enregistrement and (t > T_enregistrement)) {
      while (true) { delay(1000); } // attendre jusque la fin des temps
    }
}
```