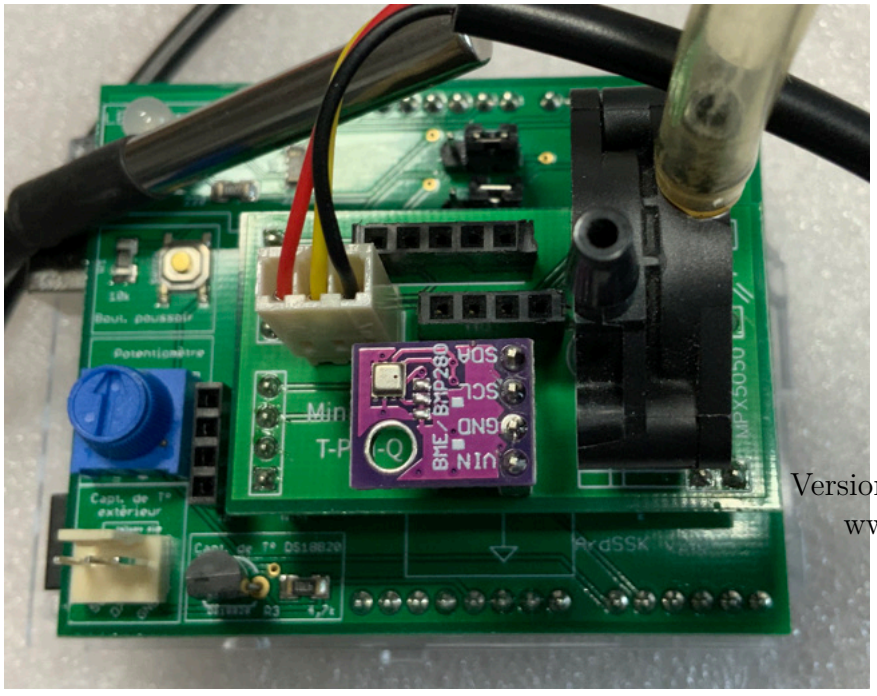
Arduino
Kit TP Sup



Version 5 octobre 2021
www.chimsoft.com

Chapitre 1

Kit TP Sup

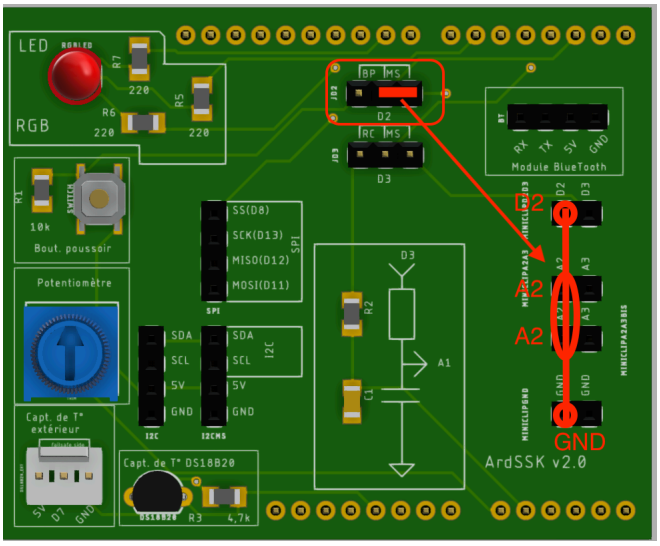
Ce kit a été conçu pour couvrir la plupart des TP, mettant en oeuvre l'utilisation d'une carte Arduino, proposés dans les nouveaux programmes de physique de sup.

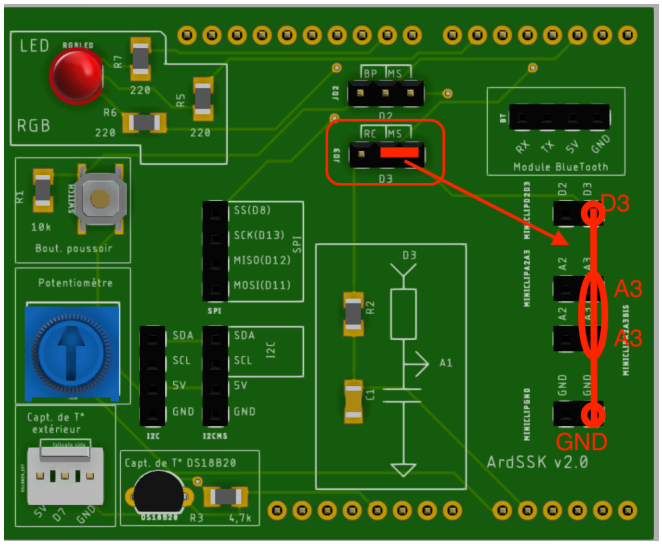
Un premier document : **ArduinoSup** décrit la prise en charge d'une carte Arduino et les premiers apprentissages à l'aide du module (shield dans le jargon arduinesque) que nous avons conçu. Ce module d'apprentissage (ArdSSK pour Arduino Scientific Starter Kit !) venait ce fixer sur une carte Arduino classique. Ce module contient un connecteur sur lequel nous pouvons fixer d'autres modules...
Nous proposons dans ce kit l'ajout possible :

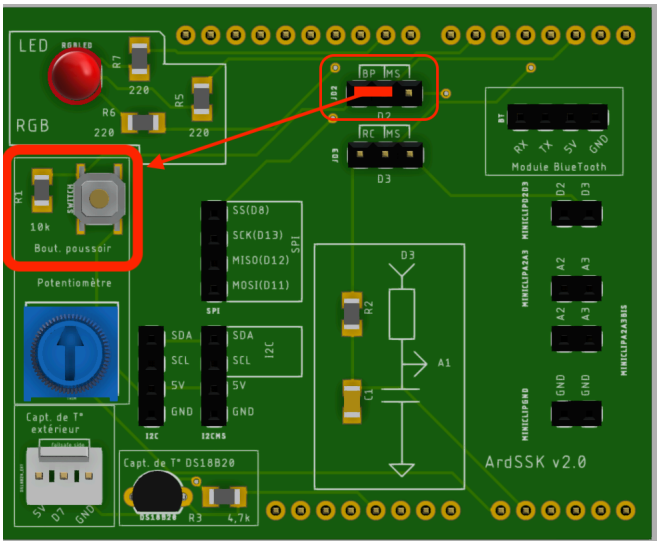
- d'un « mini clip » diviseur de tension : photorésistance ou thermistance ;
- d'un « Minishield Température » ;
- d'un « Minishield Pression » ;
- d'un « Minishield 9DOF » : accéléromètre - gyroscope - magnétomètre.

.

Certains ports de la carte Arduino (`D2`, `D3`) sont partagés entre le module d'apprentissage et le minishield. Il faut donc veiller à placer les cavaliers en bonne position. Pour l'utilisation des "mini clip" et minishields, les cavaliers doivent être positionnés sur la droite comme sur les figures ci-dessous.







Ponctuellement, vous n'aurez pas besoin du port `D2` sur le minishield, si vous souhaitez disposer de la programmation du bouton poussoir, il vous suffira de placer le cavalier vers la gauche comme sur la figure ci-contre.

On se reportera à l'annexe « Installation des bibliothèques » (page 21) pour, comme son nom l'indique, installer les bibliothèques nécessaires.

Dans une utilisation réseau au lycée, il faudra s'assurer que ces bibliothèques sont bien accessibles à partir des profils élèves si l'on souhaite leur faire charger ou modifier les programmes.

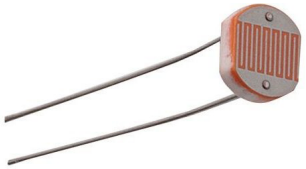
Chapitre 2

Capteur résistif

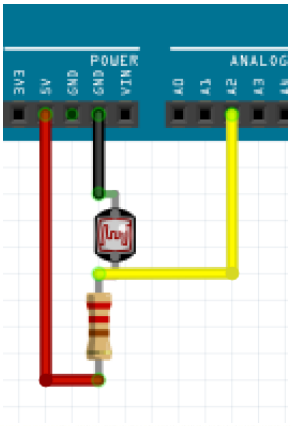
Un grand classique également des kit Arduino. L'étude d'un tel capteur est au programme dès la classe de première. !

On branche la résistance variable (photorésistance ou thermistance) en série avec une résistance de protection (typiquement 1 à 10 kΩ) afin de créer un diviseur de tension. On mesure la tension aux bornes de la photorésistance ou de la thermistance à l'aide d'une entrée analogique.

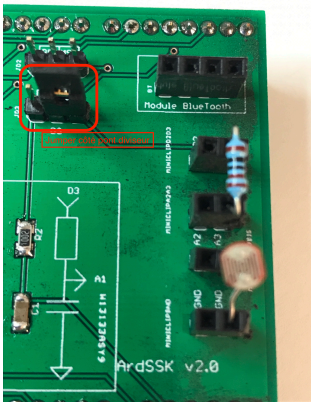
1 Une photorésistance

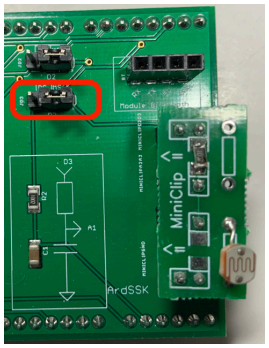


Une première solution consiste à câbler sur la platine d'essai le pont diviseur entre la broche 5V et la masse.



Une seconde solution consiste à utiliser le pont diviseur câblé sur la partie droite du module ArdSSK. Il faut penser à mettre le cavalier **D3** côté pont diviseur. Avec le montage sur la photo, la lecture analogique se fait sur le port **A3**. Ce montage permet, par exemple en TP, une étude du capteur en minimisant les risques !

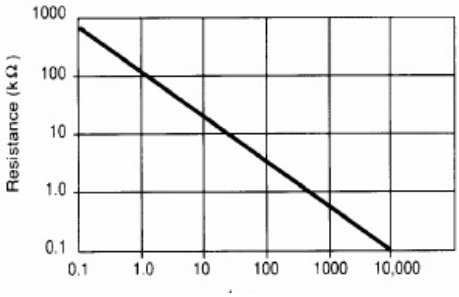




Ces risques sont encore minimisés si vous utilisez une mini carte fournie dans le kit qui vient se clipser sur les connecteurs `D3-A3-A3-GND`.
La résistance soudée sur le diviseur de tension entre **D3**(alimentation) et **A3** est une résistance de 10 kΩ.

Une mini carte vierge est également fournie afin de réaliser votre propre système.

La résistance de la photorésistance diminue quand l'intensité lumineuse augmente.



Le programme minimal d'utilisation est alors très simple.

Listing 2.1 – Programme MiniClip_LDR

```
1  #define sensorPin   A3
2  #define alimPin 3
3
4  void setup() {
5    Serial.begin(115200) ;
6    pinMode(alimPin, OUTPUT);
7    digitalWrite(alimPin, HIGH) ;
8  }
9
10 void loop() {
11   int sensorValue = analogRead(sensorPin);
12   Serial.println(sensorValue) ;
13   delay(1000) ;
14 }
```

2 Une thermistance

La thermistance proposée est de type MF52-B-3950.
C'est une résistance CTN (ou NTC en anglais) à coefficient négatif de température. A 25 ° C, la valeur de la résistance vaut $R_{25} = 10k\Omega$ (à 5% près) et la résistance R_T diminue quand la température augmente.

Température (° C)	0	10	15	20	25	30	35
R (kΩ)	98,96	20,00	15,76	12,51	10,00	8,05	6,52

L'équation de Steinhart-Hart permet de lier la température absolue T à la résistance R_T.
Sur une plage limitée de température autour de 25 ° C, on peut utiliser la formule :
$\frac{R_T}{R_{25}} = \exp\left(B \times \left(\frac{1}{T} - \frac{1}{T_{25}}\right)\right)$. Pour notre capteur, la valeur de B vaut 3950 K.

Partant du programme `MiniClip_LDR`, il « suffit » de convertir la lecture analogique en température à l'aide de la formule ci-dessus.

Listing 2.2 – Programme MiniClip_T

```
1  #define sensorPin   A3
2  #define alimPin 3
3
4  float R0 = 10000 ; // résistance du pont diviseur
5  float R25 = 10000; // résistance de référence de la
       thermistance
6  float B = 3950 ; // coefficient de température de la
       thermistance
7
8  void setup()
9  {
10   Serial.begin(115200) ;
11   sensors.begin();
12   pinMode(alimPin, OUTPUT);
13   digitalWrite(alimPin, HIGH) ;
14 }
15
16 void loop()
17 {
18   Serial.println(getT()) ;
19   delay(1000) ;
20 }
21
22 float getT()
23 {
24   int sensorValue = analogRead(sensorPin);
25   float RT=R0/((1023.0/sensorValue)-1) ;
26   float T = 1/(log(RT/R25)/B+1/298.15);
27   return T-273.15 ;
28 }
```

La routine intéressante est `getT()` :

- $sensorValue$ correspond à la lecture analogique (comprise entre 0 et 1023) ;
- la formule du diviseur de tension s'écrit : $\frac{u(lu)}{u(5V)} = \frac{R_T}{R_0 + R_T}$; on en déduit l'expression de R_T ;
- reste ensuite à calculer la température.

Le programme Thermistance (dans le dossier programme) permet d'afficher la température mesurée par la thermistance ainsi que celle donnée par le capteur DS18B20 (cablé sur la platine d'expérimentation)... capteur réputé être plus précis qu'une thermistance.

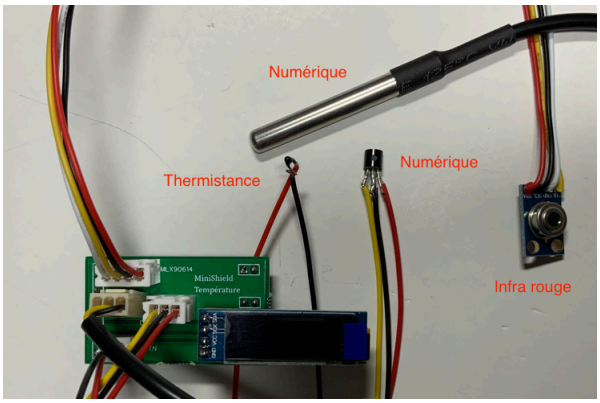
Chapitre 3

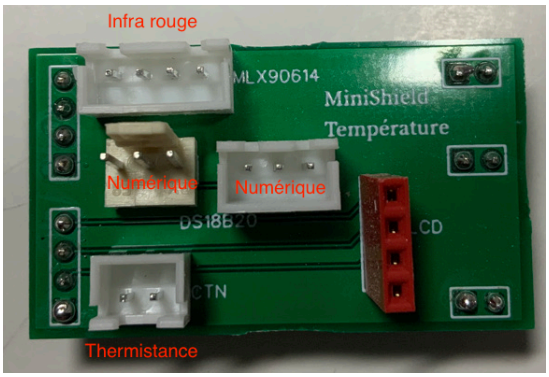
Mini Shield Température

1 Contenu du kit...

1.1 Matériel

Le « MiniShield Température » que nous proposons vient se clipser sur la carte d'apprentissage.





Soudés sur la carte se trouve :

- un connecteur (4 pins) pour capteur de température à infra rouge MLX90614 ;
- deux connecteurs (3 pins) pour capteurs de température DS18B20 ;
- un diviseur de tension avec un connecteur (2 pins) pour une thermistance ;
- un connecteur pour écran LCD.

L'un des capteurs DS18B20 est commercialisé dans une gaine étanche.
Le second a été soudé puis on a collé les connecteurs avec de l'araldite afin d'assurer une étanchéité de la connexion, tout en laissant le capteur "nu". Cette étanchéité permet de plonger ce capteur dans l'eau.
Un même traitement a été réalisé avec la thermistance.

1.2 Supports

a Datasheet

Vous trouverez dans le dossier `Datasheet` les donnés constructeurs sur ces capteurs :

- capteur infra rouge **MLX90614** : précision $\pm 0.5°$ et une résolution jusqu'à 0,02° (les afficionados veront s'il faut diviser par $\sqrt{3}$ ou autre, voire pourquoi pas par la racine carrée de la tension d'alimentation de l'arduino !) Ce capteur utilise le protocole I2C pour communiquer.
- capteur numérique **DS18B20** : précision $\pm 0.5°$ et une résolution jusqu'à 0,0625° (configuration 12 bits de résolution). Ce capteur utilise le protocole OneWire pour communiquer.

b Bibliothèques

La bibliothèque permettant d'exploiter le capteur de température infra rouge MLX90614 est celle proposée par Adafruit : `Adafruit MLX90614 Library`.

Pour le capteur de température DS18B20, les bibliothèques `OneWire` et `DallasTemperature` sont nécessaires.

Pour l'utilisation de l'écran LCD, on se reportera à l'annexe (pages 21 et **??**).

c Programme

Dans le dossier `Programmes` qui accompagne également ce kit vous trouverez le programme complet **MiniShield_Temperature** qui met en jeu ces capteurs.

2 Utilisation...

2.1 Connexion

Le module vient se clipser sur les connecteurs de la platine ArdSSK.

PLACER LES CAVALIERS D2 ET D3 COTE MS (A DROITE)

L'écran LCD peut être placé sur le minishield.

2.2 Programme de test

Charger le programme `MiniShield_Temperature` :

```
1     \item sur l'écran LCD doivent apparaitre les 2
         températures données par les 2 capteurs numériques,
         puis la température donnée par la thermistance et enfin
         celle donnée par le capteur infrarouge ;
2     \ dans le moniteur série (115200 baud), vous devriez
         obtenir (aux valeurs numériques près) la sortie
         suivante :
```

```
...
DATA:TIMES:7.8:T1:21.8:T2:21.8:CTN:21.2:IR:21.6
DATA:TIMES:9.8:T1:21.8:T2:21.8:CTN:21.2:IR:22.0
DATA:TIMES:11.7:T1:21.8:T2:21.8:CTN:21.2:IR:22.2
...
```

Toucher les différents capteurs doit avoir des conséquences sur l'affichage !

2.3 Programme MiniShield_Temperature

Listing 3.1 – Programme MiniShield_Temperature

```
1 // capteur DS18B20
2 #include <OneWire.h>
3 #include <DallasTemperature.h>
```

```
4  #define ONE_WIRE_BUS 2
5  OneWire oneWire(ONE_WIRE_BUS);
6  DallasTemperature sensors(&oneWire);
7  int nbTemperature ;
8
9  // capteur infrarouge
10 #include <Adafruit_MLX90614.h>
11 Adafruit_MLX90614 mlx = Adafruit_MLX90614();
12
13 // thermistance
14 #define sensorPin   A3
15 #define alimPin 3
16 float R0 = 10000 ; // résistance du pont diviseur
17 float R25 = 10000; // résistance de référence de la
      thermistance
18 float B = 3950 ; // coefficient de température de la
      thermistance
19
20 // affichage sur écran LCD
21 #include <Adafruit_SSD1306.h>
22 Adafruit_SSD1306 lcd(4);
23
24 void setup()
25 {
26   Serial.begin(115200) ;
27   sensors.begin();
28   nbTemperature = sensors.getDeviceCount() ;
29   Serial.println(nbTemperature) ;
30   pinMode(alimPin, OUTPUT);
31   digitalWrite(alimPin, HIGH) ;
32
33   mlx.begin();
34   Serial.println("Programme : MiniShield_Temperature") ;
35
36   lcd.begin(SSD1306_SWITCHCAPVCC, 0x3C);
37   lcd.setTextSize(1);
38   lcd.setTextColor(WHITE);
39   lcd.clearDisplay() ;
40   lcd.setCursor(0,0);
41   lcd.print("MiniShield T") ;
42   lcd.display()  ;
43 }
44
45 void loop()
46 {
47   String msg = "DATA:TIMES:" + String(millis()/1000.0,1) ;
48   lcd.clearDisplay();
49   if (nbTemperature > 0)
50   {
51     sensors.requestTemperatures(); // Send the command to get
        temperatures
52     for (int i = 0 ; i < nbTemperature ; i++)
53     {
```

```
54        msg = msg + ":T" + String(i+1) + ":" +
              String(sensors.getTempCByIndex(i),1);
55        lcd.setCursor(30*i,0) ;
56        lcd.print(String(sensors.getTempCByIndex(i),1)) ;
57      }
58    }
59    float T2 = getT() ;
60    msg = msg + ":CTN:"+String(T2,1) ;
61    float TIR = mlx.readObjectTempC();
62    msg = msg + ":IR:"+String(TIR,1) ;
63    Serial.println(msg) ;
64
65    lcd.setCursor(0,12);
66    lcd.print("T2 = ") ;
67    lcd.println(String(T2,1)) ;
68    lcd.setCursor(0,24);
69    lcd.print("T(IR) = ") ;
70    lcd.println(String(TIR,1)) ;
71    lcd.display() ;
72
73    delay(1000) ;
74
75  }
76
77  float getT()
78  {
79    int sensorValue = analogRead(sensorPin);
80    float RT=R0/((1023.0/sensorValue)-1) ;
81    float T = 1/(log(RT/R25)/B+1/298.15);
82    return T-273.15 ;
83  }
```

a Utilisation du capteur numérique DS18B20

L'importation des bibliothèques se fait lignes 2 et 3.
C'est le port **D2** qui est utilisé sur le MiniShield pour la connection `OneWire`. La gestion de ce bus est entièrement gérée par la bibliothèque.
La variable `oneWire` instanciée ligne 5 ne sert qu'à initialiser la variable `sensors` qui nous sera utile par la suite.
On peut connecter plusieurs capteurs de ce type sur le même bus, on stocke dans la variable `nbTemperature` le nombre de capteurs détectés.

Dans la fonction **setup**, on détermine effectivement ligne 28 le nombre de capteurs utilisés.
Chaque capteur possède un propre identificateur ; on ne s'en soucie pas ici mais quelques programmes que l'on peut trouver sur Internet utilisent ces identificateurs.
Dans la fonction **loop**, le code de la gestion de l'acquisition apparait lignes 49 à 58. La ligne 51 permet de "récupérer" les informations nécessaires à partir des capteurs. L'ordre d'apparition des informations est, heureusement, toujours le même !
On accède à la température en Celcius par l'instruction `sensors.getTempCByIndex(i)`. Si un seul capteur est présent, il faudra conserver les deux lignes :

```
...
sensors.requestTemperatures() ;
float T = sensors.getTempCByIndex(0) ;
```

b Utilisation du capteur infrarouge MLX90614

La bibliothèque est importée ligne 10 et la variable `mlx` instanciée ligne 11 permet de gérer l'acquisition.

Dans la fonction **setup**, ligne 33, on initialise toute la gestion sur le port I2C correspondant.

L'acquisition se fait très simplement ligne 61 à l'aide de l'instruction `mlx.readObjectTempC()`.

c Utilisation de la thermistance

C'est la même thermistance que celle décrite au chapitre précédent. Il suffit de se reporter.

d Affichage sur écran LCD

Après importation de la bibliothèque ligne 21, la variable `lcd` définie ligne 22 permet de gérer, par la suite, tout l'affichage sur l'écran de type SSD1306.

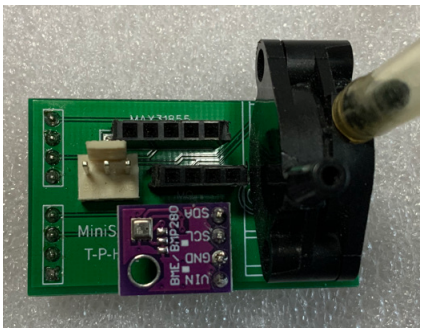
Chapitre 4

Mini Shield Pression

1 Contenu du kit...

1.1 Matériel

Le « MiniShield Pression fg que nous proposons vient se clipser sur la carte ArdSSK.



Soudés sur la carte se trouve :

- un capteur différentiel de pression MPX5500DP ;
- un connecteur pour la module BMP280 (température-pression) ;
- un connecteur pour écran LCD.

1.2 Supports

a Datasheet

Vous trouverez dans le dossier `Datasheet` les donnés constructeurs sur ces capteurs :

- capteur différentiel de pression MPX5500DP ;
- capteur digital de pression BMP280

b Bibliothèques

Aucune bibliothèque n'est nécessaire pour le capteur différentielle de pression puisque celui-ci propose une sortie analogique.

La bibliothèque permettant d'exploiter le module BMP280 est celle proposée par Adafruit.

Pour l'utilisation de l'écran LCD, on se reportera à l'annexe (pages 21 et **??**).

c Programmes

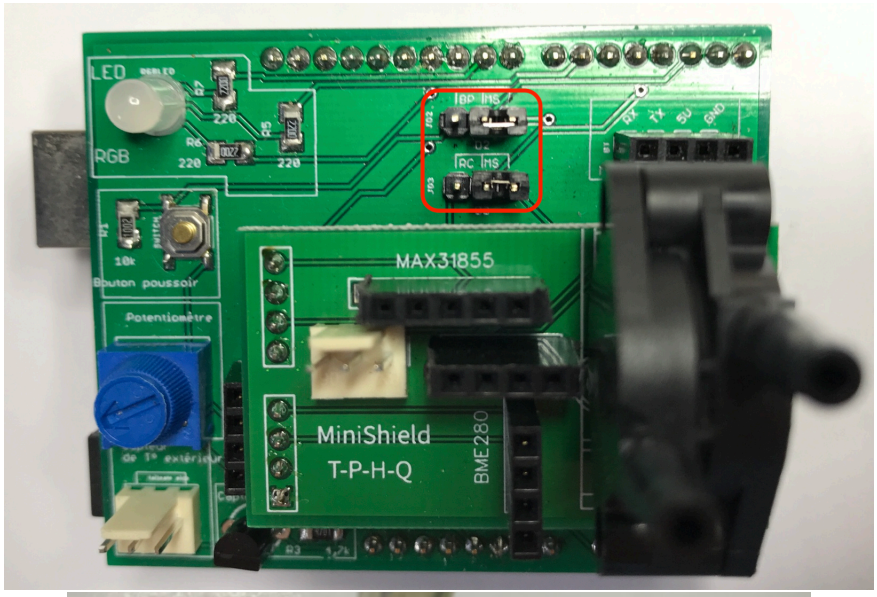
Dans le dossier `Programmes` qui accompagne également ce kit vous trouverez le programme complet **MiniShield_Pression** qui met en jeu ces capteurs.

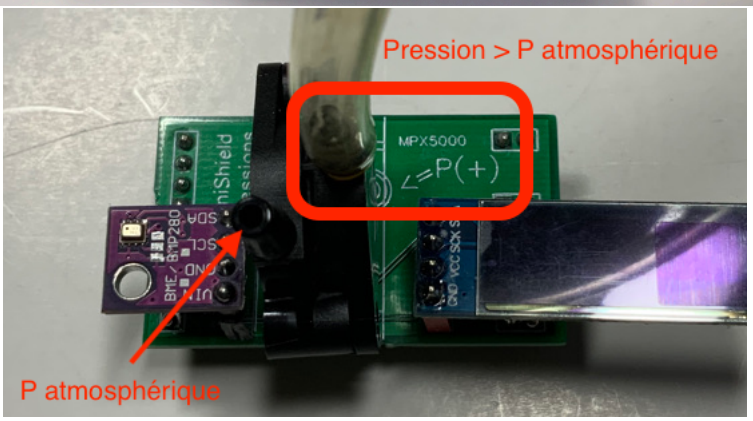
2 Utilisation. . .

2.1 Connexion

Le module vient se clipser sur les connecteurs de la platine ArdSSK.

PLACER LES CAVALIERS D2 ET D3 COTE MS (A DROITE)





MESURER LA PRESSION SUR L'EMBOUT EN FACE DE LA FLECHE P(+)

2.2 Programme de test

Charger le programme `MiniShield_Pression`, dans le moniteur série (115200 baud), vous devriez obtenir (aux valeurs numériques près) la sortie suivante :

. . .
. . .

2.3 Programme MiniShield_Pression

Listing 4.1 – Programme MiniShield_Pression

```
1 #include <Wire.h>
2 #include <Adafruit_BMP280.h>
3 #include <Adafruit_SSD1306.h>
```

```
4
5  // variables spécifiques au système étudié
6  Adafruit_SSD1306 lcd(4);
7  Adafruit_BMP280 bmp;
8
9  #define analogPin   A2
10 // fonction de transfert MPX5500  : Vout = Vs(Ha.P + Hb) avec
       P en kPa
11 #define Ha 0.0018   // à modifier si besoin
12 #define Hb 0.04     // à modifier si besoin
13 float offset = -26 ;   // à modifier si besoin => pour avoir 0
       quand deltaP = 0 !
14
15 // variables utilisées pour tous les programmes :
16 unsigned long last = 0 ;
17 long delai = 1000 ; // Delai en ms entre 2 appels de la
       fonction job
18
19 void setup()
20 {
21   Serial.begin(115200) ;
22   if (!bmp.begin())
23   {
24     Serial.println("INFO:Problème avec capteur BMEP80") ;
25   }
26   Serial.println("MiniShield Pression") ;
27   lcd.begin(SSD1306_SWITCHCAPVCC, 0x3C);
28   lcd.setTextSize(1);
29   lcd.setTextColor(WHITE);
30   lcd.clearDisplay() ;
31   lcd.setCursor(0,0);
32   lcd.print("MiniShield P") ;
33   lcd.display()  ;
34 }
35
36 void loop()
37 {
38   if (millis() - last > delai)
39   {
40     int sensorValue = analogRead(analogPin);
41     float p = 10*(((sensorValue / 1023.0) - Hb) / Ha) - offset
         ; // en hPa
42
43     String msg = "DATA:TIME(s):" + String(millis()/1000.0,0) ;
44     msg = msg + ":T:" + String(bmp.readTemperature(),1) +
         ":P(Pa):" + String(bmp.readPressure(),0)  ;
45     msg = msg + ":deltaP(hPa):" + String(p,0);
46     Serial.println(msg) ;
47
48     lcd.clearDisplay();
49     lcd.setCursor(0,0);
50     lcd.print("T = ") ;
51     lcd.println(String(bmp.readTemperature(),1)) ;
52     lcd.setCursor(0,12);
```

```
53      lcd.print("P(Pa) = ") ;
54      lcd.println(String(bmp.readPressure(),0)) ;
55      lcd.setCursor(0,24);
56      lcd.print("dP(hPa) = ") ;
57      lcd.println(String(p,0)) ;
58      lcd.display() ;
59
60      last = millis() ;
61    }
62  }
```

3 Capteurs

3.1 Capteur absolu de pression : BMP280

Il s'agit d'un module permettant l'acquisition de la température et de la pression atmosphérique. Il se connecte soit par bus I2C (utilisé ici), soit par SPI. D'après le Datasheet du constructeur, ce composant permet de mesurer :

- la température entre 0 et 65 °C (à 0,5 degré près) ;
- la pression entre 300 et 1100 hPa (à 1 hPa près).

Quelques applications de ce capteur sont décrites dans le guide ARDSSK du kit de TP complet. Rappelons, ici, une utilisation possible en altimètre. La bibliothèque utilisée fournit deux fonctions :

- `readAltitude(seaLevel)` où *seaLevel* est la pression ramenée au niveau de la mer ;
- `seaLevelForAltitude(altitude, pressionAtmospherique)` qui calcule la pression ramenée au niveau de la mer à partir de l'altitude du lieu et de la pression atmosphérique mesurée.

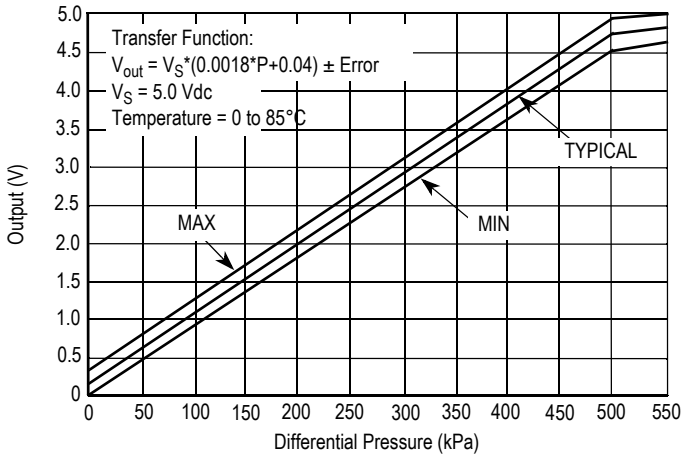
Une fois connue, à un instant donné, la pression ramenée au niveau de la mer, on peut alors se servir du capteur comme altimètre : $z = 44330 \times \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$. Dans ces conditions, une variation de la pression de 1 hPa, correspond à une variation d'altitude de 10 m.

En pratique... cette fonction donne plutôt de bon résultat. En montant 2 étages, on constate une diminution de la pression et une augmentation de l'altitude de 5 m !

3.2 Capteur différentiel de pression : MPX5500

D'après le datasheet, il s'agit d'un capteur permettant de mesurer une différence de pression de 5 bars avec une erreur maximum de 2,5 %.

Le constructeur donne l'information suivante :



Le convertisseur analogique numérique de l'Arduino fonctionnant sur 10 bits, a, quant à lui, une résolution de 5 mV. 5 bars correspond environ à 5 V ; on peut ainsi espérer une résolution de 5 milli bar soit 5 hPa.

Un décalage d'une vingtaine d'hPa est quasi systématique. Une variable $offset$ est introduite dans le programme afin d'afficher une valeur quasi nulle de l'écart de pression lorsqu'aucune différence de pression n'est appliquée. D'après ce qui précède, une oscillation de la valeur lue autour de plus ou moins 5 hPa ne doit pas surprendre.

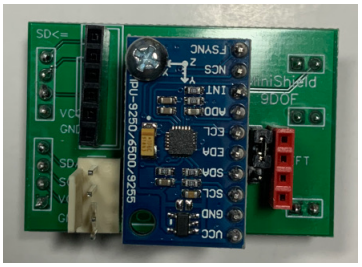
Sur le MiniShield, la lecture analogique se fait sur le port **D2** (d'où la nécessité de placer le cavalier sur la droite).

Chapitre 5

Mini Shield Accéléromètre - Gyroscope - Magnétomètre

1 Contenu du kit...

1.1 Matériel



Le « MiniShield 9DOF » que nous proposons vient se clipser sur la carte ArdSSK.

Soudés sur la carte se trouve :

- un connecteur pour le module MPU-9250/6500, le capteur n'est pas soudé directement sur la carte afin de pouvoir être utilisé éventuellement par ailleurs (un support permet de le maintenir fermement sur la carte) ;
- un connecteur (blanc 4 pins) pour connecter un second module ou ce même module à l'aide de fils souples ;
- un connecteur (3 pins mâles) + cavalier pour changer l'adresse du module si on souhaite en utiliser deux simultanément ;
- un connecteur (6 ports) pour ajouter un lecteur de carte SD ;
- un capteur (rouge 4 ports) pour écran LCD.

1.2 Supports

a Datasheet

Vous trouverez dans le dossier `Datasheet` les donnés constructeurs sur le capteur MPU9250.

b Bibliothèques

Il existe de nombreuses bibliothèques pour ce capteur. Celle proposée par Adafruit est tout à fait fonctionnelle mais occupe quasiment tout l'espace mémoire. La bibliothèque que je propose est `MPU9250_WE` dont l'utilisation, ainsi que celle du capteur est décrite sur le site : `https://wolles-elektronikkiste.de/en/mpu9250-9-axis-sensor-module-part-1`.

c Programmes

Dans le dossier `Programmes` qui accompagne également ce kit vous trouverez :

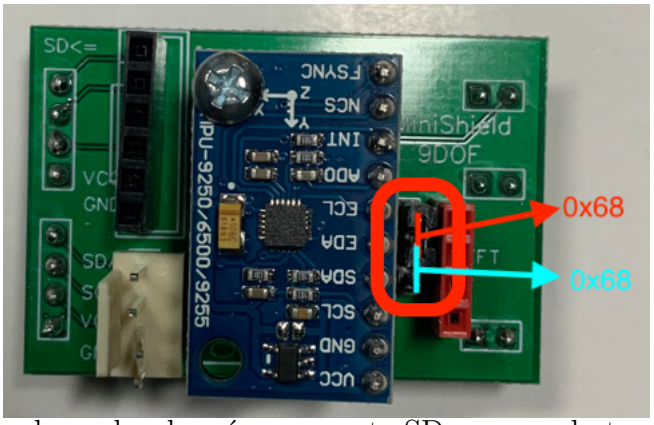
- programme `MiniShield_Accelerometre`, programme complet exploitant les 9 sorties du capteur ;
- programme `MiniShield_Accelerometre_Double` qui montre comment utiliser, simultanément 2 accéléromètres.

2 Utilisation...

2.1 Connexion

Le module vient se clipser sur les connecteurs de la platine ArdSSK.

Par défaut, l'adresse du port I2C du MPU_9250 est 0x68 (port AD0 laissé non connecté ou à la masse). Afin de pouvoir éventuellement utiliser deux capteurs, vous pouvez modifier l'adresse en plaçant le cavalier vers le bas (port AD0 connecté à 5V). Dans ces conditions, l'adresse devient 0x69.



On se reportera à l'annexe ci-après pour le stockage des données sur carte SD ou pour la transformation en accéléromètre bluetooth.

2.2 Programme de test

Charger le programme `MiniShield_Accelerometre`, dans le moniteur série (11520 baud), insérer l'écran LCD...

Pendant l'initialisation, il faut laisser l'ensemble immobile sur une surface horizontale (axe z dirigé vers le haut dans ces conditions). Ensuite, en bougeant la carte, l'accélération mesurée doit varier soit sur l'écran LCD, soit au niveau du moniteur série.

3 Programmes

3.1 Programme MiniShield_Accelerometre

Listing 5.1 – Extraits du programme MiniShield_Accelerometre

```
1  #include <SPI.h>
2
3  // capteur 9DOF : sur port I2C
4  #include <Wire.h>
5  #include <MPU9250_WE.h>
6
7  // variables spécifiques au système étudié
8  #define MPU9250_ADDR 0x68  // 0x69 si ADO connecté à VCC
9  MPU9250_WE mpu = MPU9250_WE(MPU9250_ADDR);
10 ...
11
```

```
12 void setup()
13 {
14   ...
15   mpu.init() ;
16   mpu.initMagnetometer()) ;
17   mpu.autoOffsets();
18   mpu.enableGyrDLPF();
19   mpu.setGyrDLPF(MPU9250_DLPF_6);
20   mpu.setSampleRateDivider(5);
21   mpu.setGyrRange(MPU9250_GYRO_RANGE_250);
22   mpu.setAccRange(MPU9250_ACC_RANGE_2G);
23   mpu.enableAccDLPF(true);
24   mpu.setAccDLPF(MPU9250_DLPF_6);
25   mpu.setMagOpMode(AK8963_CONT_MODE_100HZ);
26   ...
27 }
28
29 void job()
30 {
31   xyzFloat gValue = mpu.getGValues();
32   xyzFloat gyr = mpu.getGyrValues();
33   xyzFloat magValue = mpu.getMagValues();
34   float resultantG = mpu.getResultantG(gValue);
35
36   Serial.print("DATA:TIME(ms):" + String(millis() - start)) ;
37   Serial.print(":Ax:" + String(gValue.x)) ;
38   Serial.print(":Ay:" + String(gValue.y)) ;
39   Serial.print(":Az:" + String (gValue.z)) ;
40   Serial.print(":Atot:" + String (resultantG)) ;
41   Serial.print(":Gyrx:" + String(gyr.x)) ;
42   Serial.print(":Gyry:" + String(gyr.y)) ;
43   Serial.print(":Gyrz:" + String (gyr.z)) ;
44   Serial.print(":Magx:" + String(magValue.x)) ;
45   Serial.print(":Magy:" + String(magValue.y)) ;
46   Serial.print(":Magz:" + String (magValue.z)) ;
47   ...
48 }
```

Ce programme a été adapté à partir de l'exemple **MPU9250_all_data** donné avec la bibliothèque du capteur.

Ce capteur et cette bibliothèque offrent d'innombrables possibilités, y compris dans la configuration du capteur. Pour comprendre les possibilité offertes lors de l'initialisation, il faut se reporter soit au site précédemment cité qui décrit l'utilisation de la bibliothèque, soit aux riches exemples proposés.

En fonctionnement « boîte noire », j'ai recopié dans ce code les différents paramétrages proposés.

Dans ces conditions, tout est dans l'expression **xyzFloat gValue = mpu.getGValues()** qui stocke dans un tableau de 3 valeurs les accélérations sur les trois axes (en unité g), valeurs que l'on peut récupérer via **gValue.x**, y, ou z.

Il en va de même pour les données captées par le gyroscope ou le magnétomètre.

De nombreuses autres grandeurs sont accessibles et sortent du cadre de cette introduction : pitch, roll, quaternion...

3.2 Double acquisition

Il est possible de connecter 2 capteurs de ce type sur la carte. Il suffit de changer l'adresse de l'un d'eux en reliant la sortie AD0 à VCC.

Le programme `MiniShield_Accelerometre_Double` gère l'acquisition des deux mesures d'accélération.

Listing 5.2 – Extraits du programme MiniShield_Accelerometre_Double

```
1
2  #include <MPU9250_WE.h>
3
4  MPU9250_WE mpu1 = MPU9250_WE(0x68);
5  MPU9250_WE mpu2 = MPU9250_WE(0x69);
6
7
8    void setup()
9    {
10     ...
11     mpu1.init() ;
12     mpu2.init() ;
13
14     mpu1.autoOffsets();
15     mpu1.setAccRange(MPU9250_ACC_RANGE_2G);
16     mpu1.enableAccDLPF(true);
17     mpu1.setAccDLPF(MPU9250_DLPF_6);
18
19     mpu2.autoOffsets();
20     mpu2.setAccRange(MPU9250_ACC_RANGE_2G);
21     mpu2.enableAccDLPF(true);
22     mpu2.setAccDLPF(MPU9250_DLPF_6);
23     ...
24   }
25
26
27   void job()
28   {
29     xyzFloat gValue1 = mpu1.getGValues();
30     xyzFloat gValue2 = mpu2.getGValues();
31     ...
32   }
```

Il suffit « juste » d'initialiser deux variables `mpu1` et `mpu2` avec deux adresses différentes et... le tour est joué !

Chapitre 6

Annexes

1 Installation des bibliothèques

1.1 Installation « officielle »

1.2 Alternatives...

a Bibliothèques à placer dans le dossier « Library »

b Bibliothèque à placer au niveau de chaque programme

2 Éléments de syntaxe

2.1 Capteur température - humidité - pression

a Importation de bibliothèque

```
#include <Wire.h>
#include <Adafruit_BMP280.h>
```

b Instanciation

```
Adafruit_BMP280 bme;
```

c Dans la fonction setup()

```
bme.begin();
```

d Dans la fonction loop()

```
float T = bme.readTemperature();
float P = bme.readPressure(); // en Pa
```

2.2 Capteur différentiel de pression

a Importation de bibliothèque

Aucune bibliothèque à importer.

b Instanciation

#define analogPin A2 (ou int analogPin = A2 ;)

c Dans la fonction setup()

Aucun code nécessaire

d Dans la fonction loop()

int sensorValue = analogRead(analogPin) ;
float p = 10 * (((sensorValue / 1023.0) - 0.04) / 0.0018) ; // en hPa

2.3 Capteur de température

a Importation de bibliothèque

#include <OneWire.h> #include <DallasTemperature.h>

b Instanciation

OneWire oneWire(2) ; // 2 = port utilisé sur le minishield
DallasTemperature sensors(&oneWire) ;

c Dans la fonction setup()

sensors.begin() ;

d Dans la fonction loop()

sensors.requestTemperatures() ;
float T = sensors.getTempCByIndex(0) ;

3 Informations complémentaires

3.1 Trame d'un programme complet

L'intérêt principal est de pouvoir déclencher l'acquisition sur un ordre de l'utilisateur (**G**) dans le moniteur série).

Les ordres à envoyer à la carte Arduino sont les messages « traditionnels » :

- **G** : pour lancer l'acquisition ;
- **S** : pour stopper l'acquisition ;
- **Dxxx** : pour spécifier le délai en milli secondes.

4 Pour aller plus loin...

4.1 Sortie sur écran LCD

Connectez l'écran LCD sur la platine d'essai (cet écran affiche 128×32 pixels). Exécutez le programme ADC_LCD. Cette fois la valeur de la tension lue sur la broche **A0** s'affiche à l'écran.

Listing 6.1 – Programme ADC_LCD

```
1  #include <Adafruit_GFX.h>
2  #include <Adafruit_SSD1306.h>
3
4  #define analogPin A0
5  #define OLED_RESET 4
6  Adafruit_SSD1306 lcd(OLED_RESET);
7
8  void setup()
9  {
10   lcd.begin(SSD1306_SWITCHCAPVCC, 0x3C);
11   lcd.setTextSize(2);
12   lcd.setTextColor(WHITE);
13   lcd.clearDisplay() ;
14   lcd.display();
15 }
16
17 void loop()
18 {
19   int sensorValue = analogRead(analogPin);
20   int ddp = map(sensorValue, 0, 1023, 0, 5000) ;
21   lcd.clearDisplay();
22   lcd.setCursor(0,0);
23   lcd.print("mV = ") ;
24   lcd.println(ddp) ;
25   lcd.display() ;
26   delay(100) ;
27 }
```

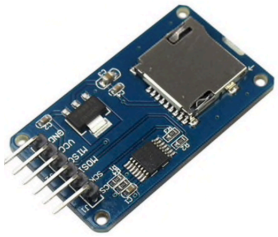
Nous utilisons, pour ce faire, une bibliothèque externe. On se contente, ici, de recopier le code de l'exemple fournit avec la bibliothèque. On instancie un objet **lcd** ligne 6. L'initialisation se fait dans la fonction `setup`. La valeur anologique est lue (ligne 19) et on effectue une conversion entre 0 et 5000 mV. L'affichage est ensuite mis à jour. L'instruction **setCursor** (ligne 22) permet de préciser le coin supérieur gauche du premier caractère (on aurait écrit (0,16) pour obtenir le texte sur la seconde ligne).

4.2 Un lecteur de carte SD pour enregistrer vos données

Le problème de stockage de données ne se pose pas pour un système connecté directement à l'ordinateur ou susceptible de communiquer par Bluetooth par exemple. Il suffit de récupérer via la communication série ces données. Charge alors à l'ordinateur (ou au smartphone) de gérer ces données.

Si on dispose d'un système autonome, ou si on souhaite disposer d'un système de sauvegarde de données « extrêmement » rapide ; l'utilisation d'une carte SD peut s'avérer intéressante. Cette utilisation est grandement facilitée par une puissante bibliothèque disponible en standard sur

Arduino.

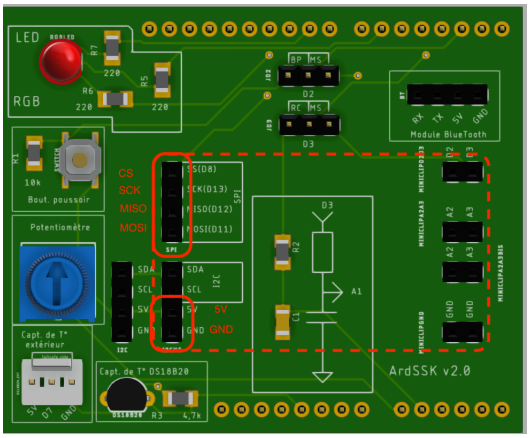


Le périphérique utilisé se connecte à l'arduino à l'aide d'un port **SPI**. Comme le bus I2C que nous avons déjà rencontré, ce bus permet une communication série avec un périphérique extérieur. Le bus SPI est bien plus rapide que le bus I2C mais un peu plus complexe à mettre en œuvre.

Six connexions sont nécessaires :

- alimentation (+ 5V ou 3,3 V) et masse ;
- une ligne d'horloge : `SCK` connectée obligatoirement à la pin `D13` de l'Arduino ;
- deux lignes de transfert `MOSI` et `MISO` respectivement connectées aux pin `D11` et `D12` de l'Arduino ;
- une ligne CS (Chip Select) que l'on connecte sur n'importe quel port numérique de l'Arduino (souvent le port 10, nous l'avons connectée sur le port `D8`) qui permet de sélectionner le périphérique I2C. Contrairement au protocole I2C, chaque périphérique aura son propre port CS permettant de l'activer.

Un bus SPI est proposé sur notre carte.
Ce dernier peut-même être utilisé sur une mini-carte de votre conception qui viendrait se clipser, en partie, justement sur le port SPI.



Connectez les différentes broches du module à l'Arduino.
L'IDE d'Arduino propose parmi ses « Exemples pour toutes cartes » quelques programmes de démonstration. Une description fort bien faite de l'utilisation de la carte SD est proposée par le `Carnet du Maker` (page : https ://www.carnetdumaker.net/articles/lire-et-ecrire-des-donnees-sur-une-carte-sd-avec-une-carte-arduino-genuino/).

Commençons par vérifier que la carte SD fonctionne bien. Pour cela, charger le programme **CardInfo**. Ce programme fait partie de l'IDE d'Arduino : menu `Fichier` puis `Exemple` puis, dans la catégorie `Exemples pour toutes cartes` sélectionner le sous-menu `SD` et enfin le programme souhaité.

Si tout se passe bien, à l'exécution, vous devez voir apparaitre dans le moniteur série les principales caractéristiques de la carte SD jointe au kit de développement. Il s'agit d'une carte SD de 120 Mo, son formatage est FAT16 et aucun fichier n'est présent sur la carte.
Vous pouvez, bien sûr, tester les autres programmes proposés.

Mais revenons à notre « cahier des charges ». On souhaite :

1. Faire l'acquisition d'une grandeur à l'aide de l'Arduino.
2. Stocker les informations correspondantes sur la carte SD.
3. Pouvoir lire ces informations ultérieurement.

Le premier point ne pose plus de problème pour nous. En anticipant un peu sur la suite, nous pouvons prendre comme programme de référence le programme `ADC_ModeleNoTimer` du chapitre

3. Tout y est... ou presque.

Nous proposons alors le programme **DataLogging_Modele**. Seules les modifications par rapport au programme d'origine sont reportées dans le listing ci-dessous.

Listing 6.2 – Programme DataLogging_Modele

```
1  ...
2  #include <SPI.h>
3  #include <SD.h>
4  ...
5  // variables spécifiques au système étudié
6  #define chipSelect  8
7  #define logFile     "data.txt"
8  File dataFile ;
9
10 void setup()
11 {
12   ...
13   if (!SD.begin(chipSelect))
14   {
15     Serial.println("INFO:Problème lecteur carte SD") ;
16   }
17 }
18
19 void affiche(String msg)
20 {
21   Serial.println(msg) ;
22   BT.println(msg) ;
23   if (dataFile)
24   dataFile.println(msg) ;
25 }
26
27 void stop()
28 {
29   doJob = false ;
30   if (dataFile)
31   dataFile.close() ;
32 }
33
34 void parse(char ordre, long valeur)
35 {
36   switch (ordre)
37   {
38     case 'A' : // append
39     dataFile = SD.open(logFile, FILE_WRITE) ;
40     break ;
41     case 'W' : // write
42     if (SD.exists(logFile))
43     SD.remove(logFile) ;
44     dataFile = SD.open(logFile, FILE_WRITE) ;
45     break ;
46     case 'R' : // read
47     read() ;
48     break ;
```

```
49        default :
50        break ;
51      }
52    }
53
54    void read()
55    {
56      dataFile = SD.open(logFile) ;
57      if (dataFile)
58      {
59        while (dataFile.available())
60        Serial.write(dataFile.read()) ;
61        dataFile.close() ;
62      }
63    }
```

Qu'a-t-on modifier ?

- Lignes 2 et 3 : on importe les bibliothèques SPI et SD.
- Lignes 6 et 7 : on précise le port ChipSelect utilisé ainsi que le nom du fichier.
- Ligne 8 : on crée une variable *dataFile* de type `File` dont le rôle est essentiel par la suite car c'est elle qui nous permettra lecture et écriture sur le fichier texte proprement dit.
- Dans le `setup`, il faut initialiser (ligne 13) la communication avec la carte SD avec le bon ChipSelect. On aurait pu utiliser un objet (comme dans le programme CardInfo par exemple).
- On ajoute quelques ordres à passer à la carte Arduino. Ligne 38, si on passe **A** (comme Append) on ouvre le fichier en écriture (s'il n'existe pas il est créé) et on écrit à la fin du fichier ; les informations déjà présentes ne sont pas perdues. Si on passe l'ordre **W** (pour Write) (ligne 41), si le fichier existe, on l'efface. On le crée à nouveau et on l'ouvre en écriture. Enfin ligne 38, si on envoie l'ordre **R** (pour Read), on appelle la fonction correspondante.
- Cette fonction `read` (lignes 46) se charge d'ouvrir le fichier en lecture, parcourir le fichier et afficher toutes les lignes une à une dans le port série.
- Ces données ont été écrites ligne 24 dans la routine d'affichage traditionnelle.
- Il faut juste penser à fermer le fichier lorsque l'on stoppe l'acquisistion (ligne 31).

Sans forcément connecter de capteur, le programme est fonctionnel. Nous verrons dans le paragraphe suivant un exercice d'application.

On peut imaginer d'autre applications à une carte SD comme, par exemple, écrire dans un fichier texte une suite d'instructions que l'on pourrait exécuter, à la demande, sur la carte.

4.3 Utilisation du module bluetooth

a Pour l'utilisation du Bluetooth

Nous allons montrer dans ce guide comment communiquer en Bluetooth avec un module Arduino. La solution la plus simple consiste à installer sur son smartphone ou tablette (iPhone ou Android) une application permettant de se connecter au module Bluetooth de la carte.

Nous proposons, par exemple, les deux applications gratuites suivantes :

- sur iPhone : l'application HM10 Bluetooth Serial Lite (https ://itunes.apple.com/fr/app/hm10-bluetooth-serial-lite/id1030454675 ?mt=8) (on fait une recherche HM10 serial sur l'Appel Store) ;

- sur Android : l'application Serial Bluetooth Terminal est une application très simple mais efficace. Sur votre téléphone, cherchez dans le *Play Store* : Serial Bluetooth Terminal.

b Millivoltmètre connecté

Dans un objet connecté, les échanges d'informations doivent pouvoir se faire avec un smartphone. Le programme LED_ModeleBT nous sert de base de départ puisque l'interface entre l'Arduino et un smartphone est déjà implémenté. La connexion à l'ordinateur est toutefois nécessaire pour la phase programmation (et éventuellement débogage).

Le code proposé du programme Voltmetre_Connecte est, espérons-le, relativement simple à comprendre maintenant.

Listing 6.3 – Programme Voltmetre_Connecte

```
#include <MsTimer2.h>
#include <SoftwareSerial.h>

// variables utilisées pour tous les programmes :
#define BAUD 9600  // vitesse d'échange pour le port série
#define Info "Voltmetre_Connecte"
long delai = 1000 ; // Delai en ms entre 2 appels de la
   fonction job
#define softTX 4
#define softRX 5
SoftwareSerial BT(softRX, softTX) ;

// variables spécifiques au système étudié
#define analogPin   A0

void setup()
{
  Serial.begin(BAUD) ;
  BT.begin(BAUD) ;
  MsTimer2::set(delai, job) ;
}

void loop()
{
  if (Serial.available())
  {
    String message = Serial.readString() ;
    parse(message) ;
  }
  if (BT.available())
  {
    String message = BT.readString() ;
    parse(message) ;
  }
}

void parse(String msg)
{
  msg.replace("\r", "") ;
  msg.replace("\n", "") ;
```

```
40      msg.replace(":", "") ;
41      char ordre = msg[0];
42      msg.replace(String(ordre), "") ;
43      int valeur = 0 ;
44      if (msg.length() > 0)
45      valeur = msg.toInt() ;
46      switch (ordre)
47      {
48        case 'I' :
49        affiche(Info) ;
50        break ;
51        case 'G' :
52        go() ;
53        break ;
54        case 'O' :
55        job() ;
56        break ;
57        case 'S' :
58        stop() ;
59        break ;
60        case 'D' :
61        if (valeur > 0)
62        {
63          delai = valeur ;
64          MsTimer2::set(delai, job) ;
65        }
66        break ;
67        default :
68        parse(ordre, valeur) ;
69        break ;
70      }
71    }
72
73    void affiche(String msg)
74    {
75      Serial.println(msg) ;
76      BT.println(msg) ;
77    }
78    // traitements spécifiques au système
79
80    void go()
81    {
82      MsTimer2::start() ;  // c'est parti !
83    }
84
85    void stop()
86    {
87      MsTimer2::stop() ; // on stoppe le timer
88    }
89
90    void parse(char ordre, long valeur)
91    {
92      switch (ordre)
93      {
```

```
94        default : // rien à faire ici, on le laisse pour le
              modèle
95        break ;
96      }
97    }
98
99    void job()
100   {
101     int sensorValue = analogRead(analogPin);
102     long ddp = map(sensorValue, 0, 1023, 0, 5000) ;
103     affiche("mV = " + String(ddp)) ;
104   }
```

Ce programme peut nous servir de modèle pour un système d'acquisition connecté. Nous y ferons référence dans le chapitre 5 pour la réalisation (et l'utilisation) d'un accéléromètre BlueTooth.

La limite, ici est que l'on ne peut pas accéder à des lectures en continu puisque le delai du timer ne peut être nul. Le programme VoltmètreContinu_Connecte gère ce cas de figure un petit peu plus complexe (comme le faisait, d'ailleurs le modèle LED_ModeleBT).