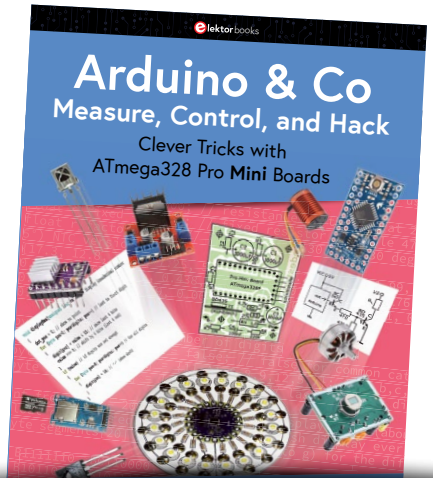
enregistreur d'infrasons avec l'Arduino Pro Mini

un projet extrait du livre « Arduino & Co. » d'Elektor

Robert Sontheimer (Allemagne)

Réalisons un projet inédit : combinons un Arduino Pro Mini (ou une carte compatible) avec un capteur de pression atmosphérique BMP180 et un module de carte SD, et étudions l'enregistrement à long terme des niveaux d'infrasons.



Note de la rédaction. *Cet article est un extrait du livre de 332 pages Arduino & Co — Measure, Control, and Hack, formaté et légèrement modifié pour correspondre aux normes éditoriales et à la mise en page du magazine Elektor. L'auteur et l'éditeur ont fait de leur mieux pour l'éviter et seront heureux de répondre aux questions – pour les contacter, voir l'encadré* **« Questions ou commentaires ? »**.

Le capteur de pression atmosphérique BMP180 délivre un maximum d'environ 100 valeurs de température et de pression atmosphérique par seconde. Nous pourrions donc générer un fichier audio stéréo (au format PCM avec l'extension .wav) à partir de ces données, sur des minutes, des heures ou des jours, en enregistrant la pression atmosphérique sur le canal gauche et la température sur le canal droit. Nous réglons la fréquence de lecture régulière (taux d'échantillonnage) sur 44 100 Hz, par exemple. Il s'agit d'une valeur courante qui est également utilisée pour les CD audio et bien d'autres choses. Ainsi, le fichier sera lu plus tard (par exemple sur un ordinateur portable ou avec un lecteur audio) à 441 fois la vitesse normale, et nous pourrons entendre des infrasons ! Avec la fréquence d'échantillonnage originale de 100 Hz lors de l'enregistrement, on peut enregistrer toutes les fréquences inférieures à 50 Hz. Un infrason de 10 Hz devient 4,41 kHz lors de la lecture. 1 Hz devient 441 Hz, etc. Théoriquement, il n'y a pas de limites en dessous.

Il est vrai que le BMP180 n'est pas vraiment adapté à un microphone très sensible. Bien qu'il détecte déjà les écarts de pression lors de faibles changements d'altitude, les infrasons doivent être assez forts pour être entendus malgré le bruit pendant la lecture. Cependant, il est beaucoup plus intéressant d'utiliser un logiciel d'édition audio pour examiner les formes d'onde de la pression atmosphérique et de la température plus tard - surtout si l'enregistrement s'est déroulé sur plusieurs jours. Le canal de gauche présente alors la courbe de la pression atmosphérique, et celui de droite, la courbe de la température.

Ici, j'ai simplement enregistré pendant près de trois jours, ouvert le fichier .wav final avec un programme d'édition audio, puis normalisé les deux canaux séparément, amplifié autant que pour que la gamme complète soit utilisée. Sur la **figure 1**, la courbe supérieure avec les relevés de pression semble encore un peu épaisse et irrégulière car chaque pixel sur l'axe X est constitué de dizaines de milliers de mesures qui contiennent également du bruit et fluctuent donc à un certain niveau.

La **figure 2** montre les résultats après traitement. L'enregistrement a été rééchantillonné et 4 000 valeurs ont été moyennées en une seule valeur. Le bruit a disparu et nous pouvons voir plus clairement les courbes de pression et de température.

Les graphes nous permettent de constater certaines choses : Sur la courbe de pression en haut, en termes de temps, il y a eu un peu plus de vent dans la deuxième moitié de la mesure, car de nombreuses petites fluctuations augmentent à ce moment-là. Lors d'une vraie tempête, les fluctuations seraient encore plus extrêmes.

La courbe de température en bas montre clairement le rythme jour-nuit. Les petits pics presque continus correspondent aux cycles de commutation du thermostat pour le chauffage de la pièce. On pourrait compter le nombre de fois que le thermostat s'est allumé et éteint au cours de ces trois jours. En outre, on constate

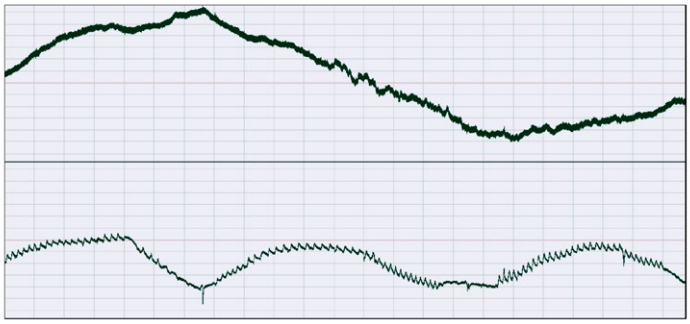


Figure 1. Progression du graphique de la pression atmosphérique (en haut) et de la température (en bas), essentiellement non traitée.

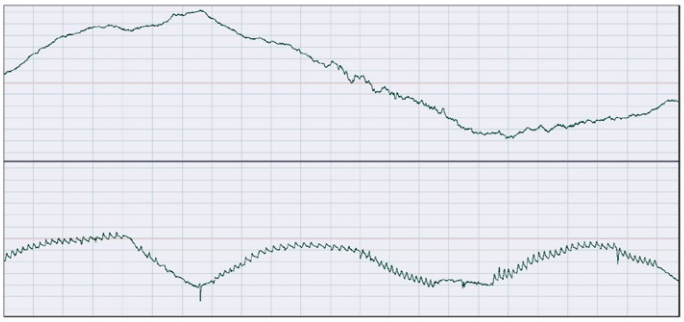


Figure 2. Évolution graphique de la pression atmosphérique (en haut) et de la température (en bas), traitée.

deux déviations rapides vers le bas : la fenêtre a été ouverte pendant une courte période.

Construction

Pour réaliser le projet, nous avons besoin de :

- 1 carte Arduino (16 MHz ATmega328)
- 1 module de capteur BMP180
- 1 module de carte (Micro)SD
- 1 bouton-poussoir
- fils ou cavaliers
- 1 carte (Micro)SD, max. 32 GB

La construction est assez simple. En plus du capteur BMP180, il faut connecter le module de carte SD, ainsi que le bouton poussoir pour démarrer et arrêter l'enregistrement. Il suffit de respecter le schéma de connexion de la **figure 3**. Les fils soudés sont toujours préférables, mais nous pouvons aussi utiliser des cavaliers. Ensuite, nous avons besoin de deux lignes VCC, mais le Pro Mini n'a qu'une seule broche VCC. La solution est très simple : puisque le BMP180 ne consomme pratiquement pas d'énergie, nous pouvons également l'alimenter avec une sortie (par exemple, avec la broche 9 au lieu de VCC). Il suffit alors de configurer la broche en sortie dans `setup()` et de la mettre à l'état haut.

Croquis de l'enregistreur d'infrasons

Une importante partie du croquis *13.9.ino* écrit pour ce projet sera présentée dans les listages numérotés **listage 1a** à **1h** ainsi que les paragraphes ci-dessous. Le programme complet est trop long pour être présenté ici avec des listages partiels. Le programme et les sous-programmes mentionnés ci-dessous sont contenus dans le fichier d'accompagnement du logiciel publié pour le livre, disponible en téléchargement gratuit sur la page d'Elektor [1]. Sur la page web, cliquez sur l'onglet *Téléchargements*. Pour pouvoir suivre cet extrait, accédez au fichier *13.9.ino*.

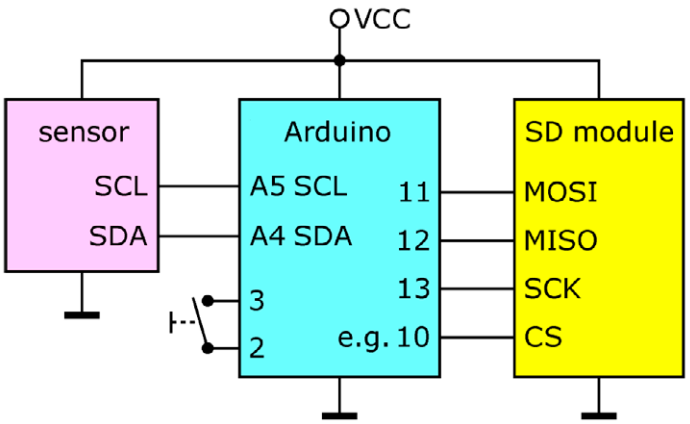


Figure 3. Schéma de connexion pour le capteur BMP180, le module de carte SD, l'interrupteur et la carte Arduino.

Listage 1a : comme nous combinons le capteur de pression atmosphérique avec le module de carte SD, nous devons inclure trois bibliothèques : l'interface I²C pour le capteur de pression atmosphérique, l'interface SPI pour le module de carte SD et les fonctions spéciales de la carte SD.

Ensuite, nous définissons deux broches où nous pouvons connecter un bouton poussoir pour démarrer et arrêter l'enregistrement. La broche 3 est commutée au niveau bas et sert de masse pour le bouton. Cette méthode est très pratique, car les deux broches du bouton sont alors très proches l'une de l'autre. Bien sûr, nous pourrions aussi utiliser la vraie masse (GND) à la place de la broche 3.

Les autres définitions et variables, ainsi que toutes les parties du programme du capteur de pression d'air et les croquis de la carte SD discutés dans le livre, ne sont pas montrés ici. Nous ne nous intéressons maintenant qu'au code relatif à l'enregistreur.

Listage 1a

```
#include <Wire.h>     // library for I2C interface
#include <SPI.h>      // library for SPI interface
#include <SD.h>       // library for SD card
#define record_pin 2 // switch pin to ground for
                      //   recording
#define low_pin 3     // Set output pin to LOW
                      //   (used as GND for button)
…
```

Listage 1b : ici, les paramètres de l'enregistreur sont présentés. La variable `oversampling` détermine également le nombre de mesures de pression. Pour enregistrer 100 valeurs par seconde, ce paramètre doit être 0. `audio_rate`, spécifie le nombre de valeurs par seconde qui seront transmises lors de la lecture ultérieure de l'enregistrement sous forme de fichier audio. `max_file_size` détermine la longueur maximale de l'enregistrement. Cependant, les 10 minutes font référence au temps de lecture – le temps d'enregistrement correspondant est de plusieurs jours. Cependant, nous pouvons modifier cette valeur de manière presque arbitraire. Lorsque la taille maximale du fichier est atteinte, le fichier est fermé et un nouvel enregistrement est lancé.

La variable `duration` détermine le délai entre les mesures. Cependant, les durées sont ici plus longues de 500 µs car nous travaillons plutôt avec une horloge fixe, et les instructions `Wire` pour la mesure de la température et de la pression sont également exécutées.

Listage 1b

```
// 0 -> 1x, 1 -> 2x, 2 -> 4x, 3->8x
char oversampling = 0;
// sampling rate of the output audio file
long audio_rate = 44100;
// 10 minutes' playback time (at 44,100 Hz)
unsigned long max_file_size = 105840044;
// time (µs) according to oversampling
int duration[4] = ;
// time for temperature measurement
int t_duration = 5000;
…
```

Listage 1c : Après les variables spéciales pour le calcul de la température et de la pression, que nous avons encore sautées ici, ces variables seront nécessaires plus tard pour l'enregistreur.

Listing 1c

```
// system time at which the data are available
unsigned int next_time;

long zero_value;  // pressure zero line
long record_value;// pressure value to save
long zero_temp;   // temperature zero line
long temp_value;  // temperature value to save

// indicates whether recording is activated
boolean must_record = false;
// indicates if recording is currently running
boolean is_recording = false;
// indicates whether pushbutton is pressed
boolean pressed = false;
// actual file size
unsigned long file_size;
// actual file number
unsigned int file_number = 0;
// counts how long button is not pressed anymore
byte state_counter = 0;
File wave_file;   // recording file
```

Listage 1d : `setup()` commence par la définition des broches du bouton-poussoir. Grâce à la résistance pull-up interne active, nous n'avons pas besoin d'autres composants.

Listing 1d

```
void setup() {
  // switch pin on input with pull-up
  pinMode(record_pin, INPUT_PULLUP);
  // low pin on output ...
  pinMode(low_pin, OUTPUT);
  // ... and LOW [...]
  digitalWrite(low_pin, LOW);
  …
}
```

Listage 1e : Dans `loop()`, nous n'avons que les appels alternés des fonctions de mesure de la température et de la pression, car pour gagner du temps, nous n'attendons pas à chaque fois les données, mais nous utilisons le temps d'attente pour exécuter les tâches suivantes. Pour la mesure de la température, on appelle la fonction `calculate()` en attendant le résultat. Pour la mesure de la pression, on appelle la fonction `recording()`.

Listing 1e

```
void loop() {
  get_t(); // read temperature
  get_p(); // read pressure
}
```

Listage 1f : Examinons la fonction `calculate()`. La température et la pression actuelles sont d'abord déterminées à partir des valeurs mesurées, si la pression a déjà été mesurée. J'ai à nouveau abrégé cette fonction par «…», car je suis sûr que personne ne souhaite relire ces calculs (de nombreuses formules de la fiche technique du BMP180).
Ensuite, nous vérifions si les références zéro pour la pression et la température sont déjà définies. Si ce n'est pas le cas, la première mesure est alors définie comme valeur de base. Si, au contraire, aucune mesure n'a encore été effectuée, on quitte complètement la fonction.
Sinon, les valeurs sont maintenant prêtes pour l'enregistrement. Pour ce faire, nous soustrayons la valeur du zéro de la pression. Si la valeur est dépassée, elle se limite à l'intervalle admissible de -32,768 à 32,767. Nous procédons ensuite exactement de la même manière pour la température. Les deux valeurs sont alors prêtes à être enregistrées.

Listage 1f

```
// calculate temperature (in tenth degree) and
// pressure (in pascals)
void calculate() {

  // if pressure has already been measured before
  if (p) {
    …
  }

  // if zero value has not been determined yet
  if (!zero_value) {
    // cancel if no measurement has been taken yet
    if (!pressure) return;
    // actual value as zero line
    zero_value = pressure;
    // actual value as zero line
    zero_temp = b5;
  }

  // pressure value for recording
  record_value = pressure - zero_value;
```

```
  // max value if clipping
  if (record_value > 32767)
    record_value = 32767;
  // negative clipping
  else if (record_value < -32768)
    record_value = -32768;
  // temperature value for recording
  temp_value = b5 - zero_temp;
  // max value if clipping
  if (temp_value > 32767) temp_value = 32767;
  // negative clipping
  else if (temp_value < -32768) temp_
  value = -32768;
}
```

Listage 1g : Dans la fonction `recording()`, deux conditions sont d'abord vérifiées : si l'enregistrement doit être exécuté (parce qu'il est commencé et qu'il n'a pas encore atteint la taille maximale du fichier) et si l'enregistrement est effectivement en cours d'exécution. Ces deux distinctions donnent lieu à quatre possibilités. Dans le premier cas, l'enregistrement devrait être en cours d'exécution, mais il ne l'est pas encore.

Listage 1g

```
void recording()  { // recording function
  // if recording should run
  if (must_record && file_size < max_file_size) {
    // if recording is not running yet
    if (!is_recording) {
      …
    }
  }
}
```

Listage 1h : Ici, un nouvel enregistrement est lancé. Le numéro de fichier est incrémenté dans la boucle do-while jusqu'à ce qu'on obtienne (avec le texte supplémentaire) un nom de fichier qui n'existe pas encore. Un nouveau fichier est alors ouvert avec le nom de fichier correspondant, et un message correspondant est également édité en série. En cas d'erreur, un autre message est émis et le processus est arrêté avec une boucle infinie.
Nous avons donc atteint la limite de l'espace alloué à la discussion sur le programme dans cet article. Heureusement, le reste du programme permettant d'utiliser l'enregistreur d'infrasons est bien documenté, tout comme les listages 1a à 1h présentées ici.

Listage 1h

```
// search for the first not yet used
// recording number
do {
  file_number++; // next number (starting with 1)
} // repeat with next number while
   //file already exists
while (SD.exists("FILE_" + String(file_number)
       + ".wav"));
Serial.println("Recording no."
               + String(file_number)
               + " is started!");
wave_file = SD.open("FILE_"
                    + String(file_number)
                    + ".wav", FILE_WRITE);

if (!wave_file) { // if file could not be opened
   Serial.println
      ("Error: File could not be opened!");
// do not continue (infinite loop)
    while (true);
}
```

Utilisation

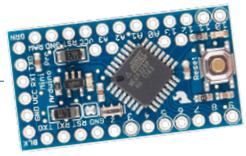
Au début du croquis, nous pouvons définir trois variables :

- `oversampling` doit être fixée à 0 pour utiliser la fréquence d'enregistrement la plus rapide de 100 Hz.
- `audio_rate` spécifie la fréquence de lecture ultérieure. On peut également utiliser une valeur inférieure à 44 100, afin que le facteur temps (44 100 / 100 = 441) ne soit pas si élevé. Les valeurs courantes sont 8 000, 11 025, 16 000, 22 050, 32 000, 44 100 et 48 000. Pas tous les lecteurs peuvent jouer d'autres fréquences.
- avec `max_file_size`, on définit la taille maximale du fichier. 105 840 044 correspond à une durée de lecture de 10 minutes à 44 100 Hz, mais à une durée d'enregistrement de plus de trois jours.

Il faut ensuite «normaliser» l'enregistrement final, c'est-à-dire augmenter le volume du signal pour utiliser toute la gamme, mais sans dépasser le maximum. Sans normalisation, le signal sera trop faible. Bien sûr, nous pourrions ajouter un facteur de gain lors de l'enregistrement, mais nous ne savons pas à l'avance à quel point la température et la pression atmosphérique varieront pendant l'enregistrement. Il est possible d'effectuer la normalisation en quelques clics avec un programme d'édition audio. Elle doit être effectuée séparément pour les deux canaux. Avec un tel programme, nous pouvons également visualiser l'enregistrement sous forme de graphiques de mesure. Le canal de gauche montre l'évolution de la pression atmosphérique pendant l'enregistrement, tandis que le canal de droite montre la température.

Enregistreur météorologique

Si nous changeons la valeur de `oversampling` à 3, seulement un 32 bonnes valeurs par seconde (au lieu de 100) seront enregistrées, ce qui est encore beaucoup plus que ce dont nous avons besoin pour les enregistrements météorologiques. Nous pourrions également combiner l'altimètre (de la section 13.8 du livre) avec les fonctions de la carte SD pour que nous puissions écrire une ligne de texte correspondante dans un fichier pour chaque sortie série également. Ainsi, nous obtenons l'enregistrement météorologique sous forme de fichier texte.



Les chapitres du livre proposent, outre des projets concrets, de nombreuses informations et suggestions qui devraient vous aider à réaliser vos propres idées.
Nous sommes arrivés à la fin de cet article, mais je vous souhaite beaucoup de plaisir dans la mesure, le contrôle et le piratage !

230393-04

Questions ou cmmentaires ?

Envoyez un courriel à l'auteur (r.sont@freenet.de) ou contactez Elektor (redaction@elektor.fr).

Produits

- **Robert Sontheimer, *Arduino & Co., Measure, Control, and Hack*, Elektor 2022**
  version papier : https://elektor.fr/20243
  version numérique : https://elektor.fr/20244

À propos de l'auteur

Robert Sontheimer a été immédiatement à bord il y a environ 40 ans, avec son ZX81 et son C64 lorsque les premiers ordinateurs familiaux sont arrivés à nos domiciles. À l'époque, il a transformé un traceur en scanner, et il a eu d'autres idées originales et bizarres. Aujourd'hui, il utilise des Arduino Pro Minis pour contrôler des machines laser CNC entières et il a même inventé un système d'aspiration adapté : son «filtre à papier hygiénique auto-changeant». Dans son bureau, il a un aimant qui lévite depuis des années - bien entendu contrôlé par un Pro Mini.

LIENS

[1] Logiciels pour le livre : https://elektor.fr/20243