de la couleur au son

Comment exploiter un capteur de couleurs avec l'I²C

Annika Schlechter (Allemagne) et
Volker Ziemann (Suède)

L'électronique peut être utile pour les personnes handicapées, comme le montre ce projet d'étudiants. Pour les personnes qui ne peuvent pas voir les couleurs, un capteur de couleurs peut assurer cette fonction : un vibreur est utilisé comme retour acoustique pour les informations de couleur et de luminosité. Dans cet article, nous présentons deux versions du projet basé sur un Arduino Nano, une carte bon marché.

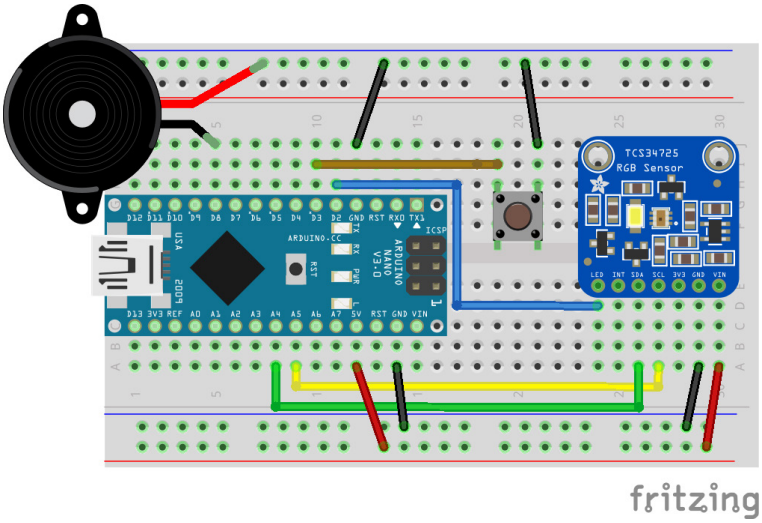


Figure 1 : câblage du prototype. Un vibreur piézoélectrique et un capteur de couleurs TCS34725 monté sur une petite carte de liaison sont connectés à un Arduino Nano.

Comment une personne malvoyante peut-elle choisir le matin la couleur de ses vêtements pour une tenue assortie ? Et bien, en convertissant l'information sur la couleur en un signal sonore. C'est l'idée de ce projet. La couleur est convertie en hauteur du son et la luminosité en durée du son.

La figure 1 montre le câblage du prototype avec un vibreur piézoélectrique et un capteur de couleurs ams TCS34725 monté sur une petite carte de liaison [1]. Les deux composants sont connectés à un Arduino Nano. Ce prototype a été mis au point dans le cadre du cours « From Sensor to Report » à l'Université d'Uppsala (Suède) [2] qui enseigne au grand public les compétences de base en acquisition de données, notamment l'interfaçage des capteurs et des microcontrôleurs. C'est pourquoi nous n'utilisons pas de bibliothèques prêtes à l'emploi pour l'interface avec le capteur, mais nous codons nous-mêmes les fonctions de base pour extraire les données des capteurs. De plus, la logique d'assignation des sons aux couleurs réside initialement dans un script Python sur un ordinateur portable, et non pas dans le microcontrôleur. Le PC et le microcontrôleur communiquent par l'intermédiaire d'une interface série. Nous avons mis au point un protocole de communication pour envoyer les valeurs du Nano vers le PC, tandis que les commandes vont en sens inverse pour activer le vibreur avec la bonne fréquence. Ensuite, nous avons amélioré le prototype pour permettre un fonctionnement autonome sur le Nano. Néanmoins, il ne s'agit pas d'un produit prêt pour une mise en production, mais d'un prototype qui met en œuvre la fonction de base et invite le lecteur à poursuivre l'expérimentation.

Avant de commencer à assembler les composants, il faut installer l'environnement de développement (EDI) Arduino [3] pour créer les programmes du Nano. L'EDI est disponible pour tous les systèmes d'exploitation fréquemment utilisés. Suivez les instructions correspondant à votre système. Si vous n'avez jamais utilisé de cartes Arduino auparavant, consultez les tutoriels Arduino [4] afin de vous familiariser avec l'EDI Arduino. Comme second prérequis, vous devez installer Python [5], également disponible pour tous les systèmes. Les fichiers binaires Windows et MAC OS sont disponibles sur le site web de Python [5] et pratiquement toutes les distributions Linux en ont des *packages* dans leurs dépôts. À nouveau, suivez les instructions pour votre système. Mais, assez de préparatifs, il est temps de commencer !

Le capteur de couleurs

Ce projet repose sur le capteur de couleurs TCS34725 [6] qui comprend un réseau de 3×4 photodiodes (**figure 2**). Les filtres photosensibles installés devant les diodes les rendent sensibles à la lumière rouge, verte et bleue. De plus, une diode claire dépourvue de filtre fournit des

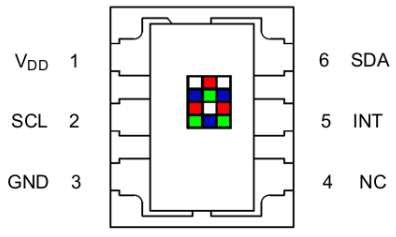


Figure 2 : le capteur de couleurs TCS34725 intègre un réseau de 3×4 photodiodes. Les filtres photosensibles installés devant les diodes rendent celles-ci sensibles à la lumière rouge, verte et bleue. Source : fiche technique / ams [6].

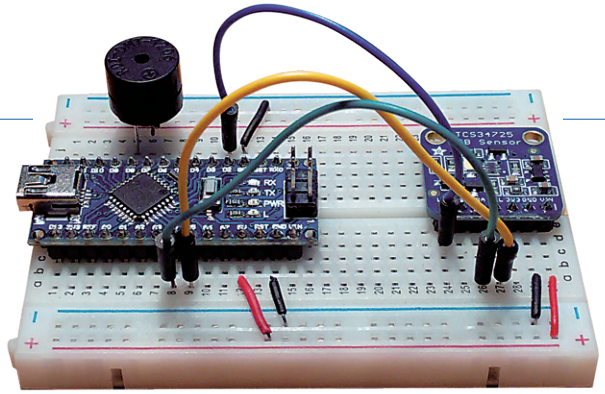


Figure 3 : le matériel (voir figure 1) est assemblé sur une platine d'expérimentation (ici sans le bouton).

PRODUITS

- **Arduino Nano**
  www.elektor.fr/arduino-nano
- **Platine d'expérimentation (830 points)**
  www.elektor.fr/breadboard-830-tie-points

informations sur la luminosité. Le signal analogique issu des diodes est converti sous forme numérique grâce aux convertisseurs analogique-numérique embarqués. L'information sur la couleur est disponible grâce à une interface I²C reliée à un microcontrôleur à l'aide uniquement de deux fils pour la communication : un pour le signal d'horloge (SLC) et un autre pour les données (SDA). Le capteur de couleurs est ensuite commandé par l'écriture et la lecture de ses registres. Mais laissons cela pour plus tard, lorsque nous décrirons le logiciel.

Le matériel est assemblé sur une platine d'expérimentation (**figure 3**), de manière assez similaire à la figure 1 avec le Nano visible du côté gauche et le capteur de couleurs TCS34725 sur la droite. Ce dernier est relié avec des fils rouges et noirs à la masse et au rail de 5 V en bas. De même, les broches correspondantes du Nano sont reliées au rail d'alimentation. Les fils verts et jaunes relient les broches de données (SDA) et d'horloge (SCL) aux broches correspondantes sur le Nano, situées en A4 et A5 respectivement. Le fil bleu qui relie la broche de sortie numérique D2 à la broche étiquetée *LED* sur le capteur de couleurs permet de commander une LED blanche montée sur la carte de liaison. Le vibreur est connecté à la broche D8 sur le Nano. Le bouton qui relie la broche D3 à la masse avec les fils marron déclenche une conversion lors le Nano fonctionne en mode autonome, c'est expliqué ci-dessous.

Ce circuit s'anime grâce à un croquis Arduino qui inclue en premier lieu la bibliothèque *wire.h* pour fournir des fonctions de bas niveau afin de communiquer dans les lignes de l'I²C. Ces fonctions sont utilisées immédiatement dans nos deux fonctions `I2Cread()` et `I2Cwrite()`, qui englobent toutes les communications avec le capteur.

La fonction `I2Cread()` sert à extraire une valeur ou un paramètre du capteur, stockés dans ce que l'on appelle des registres. Comme paramètres, la fonction reçoit l'adresse du capteur sur le bus I²C, ici `0x29` et l'adresse du registre (il nous a fallu du temps pour comprendre qu'il faut toujours lui ajouter 0x80). La fonction retourne le contenu de ce registre.

La fonction `I2Cwrite()` est utilisée pour envoyer des commandes du contrôleur vers le capteur, par exemple pour régler les valeurs de configuration. Cette fonction reçoit l'adresse du capteur et du registre ainsi que la nouvelle valeur qui doit être inscrite dans ce registre. Cette fonction ne retourne aucune valeur (`void`).

L'application

Revenons à notre croquis spécifique. Dans la fonction `setup()`, nous initialisons la communication série, les fonctions I²C de base et le capteur (par l'intermédiaire de la fonction `color_begin()`). Dans la boucle principale `loop()`, exécutée indéfiniment, nous vérifions en premier lieu si une commande est arrivée du PC sur la ligne série. Si c'est le cas, le contenu des registres du capteur est lu pour les trois couleurs. D'après la fiche technique, pour chaque couleur, il faut lire deux registres à 8 bits et assembler les octets reçus en un mot de 16 bits. Voici ce que cela donne la couleur rouge :

```
b3=I2Cread(TCS34725,0x16);   // raw data, red
b4=I2Cread(TCS34725,0x17);
red=b4*256+b3;
```

Ici `b3` contient l'octet de poids faible et `b4` l'octet de poids fort, qui est multiplié par 256 et ajouté à `b3` pour obtenir la variable à 16 bits `red`. Nous traitons les autres couleurs de la même façon ; seules les adresses des registres sont adaptées.

De plus, nous lisons la luminosité mesurée par la diode non filtrée et la stockons dans la variable `clea`. Cela déterminera la durée `d` du son. Si la valeur de `clea` est inférieure à 1000, le son durera 0,5 s ; si elle est plus élevée, ce sera 1,5 s. Ensuite la commande reçue est lue et le résultat est copié dans le tableau de caractères `line`, on peut ainsi utiliser la fonction standard du langage C `strstr()` afin de déterminer quelle commande a été envoyée. Ceci est illustré dans l'extrait de code suivant :

```
Serial.readStringUntil('\n').toCharArray(line,30);
if (strstr(line,"color?")==line) {
  Serial.print(clea); Serial.print(" ");
  Serial.print(red); Serial.print(" ");
  Serial.print(green); Serial.print(" ");
  Serial.println(blue);
} else if (strstr(line,"tone_red")==line) {
  tone(buzz, 262, d);
...
```

On peut voir que la commande *color?* entraîne le renvoi par le Nano sur la ligne série des valeurs des quatre signaux : `clea`, `red`, `green` et `blue`. Si *tone_red* est reçu, la commande intégrée `tone()` est utilisée pour activer le vibreur piézoélectrique, dans le cas présent avec la fréquence qui est assignée à la couleur rouge (*red*). Plusieurs blocs `strstr()` construits de la même façon suivent et entraînent les actions appropriées.

Notez que nous utilisons un protocole simple, basé sur des allers-retours de chaînes de caractères ; une demande de l'ordinateur portable se termine par un point d'interrogation, par exemple « *color?* » et le Nano répond en renvoyant les valeurs. Une instruction est basée sur le simple envoi de la commande, par exemple « *tone_red* » qui provoque l'émission d'un son par le vibreur à 262 Hz pendant la durée spécifiée par `d`. La communication est à peu près équivalente au langage de commande SCPI qui est pris en charge par les oscilloscopes modernes et d'autres appareils de test et de mesure. Cela facilite l'interfaçage avec des programmes externes qui prennent en charge l'accès à la ligne série ; c'est le cas de Python, Octave, Matlab et Labview.

Le programme sur PC

Nous utilisons Python 3 sur l'ordinateur portable pour communiquer avec le Nano. Le **listing 1** présente le code assez élémentaire.

Après avoir importé les bibliothèques nécessaires pour la communication série et la synchronisation, le port série sur lequel le Nano est connecté est ouvert à une vitesse de transmission (baud) correspondant à celle choisie sur le Nano. Trois secondes sont allouées au système d'exploitation pour établir la connexion, puis la commande « *color?* » est envoyée.
Notez que Python 3 code les chaînes de caractères en Unicode, tandis que le Nano attend de simples chaînes ASCII. Il faut par conséquent ajouter la lettre « b » à la commande « *color?* » afin d'envoyer la chaîne en tant que simples données ASCII. Après un autre court temps d'attente, la réponse envoyée par le Nano est stockée dans la variable `reply` qui contient les valeurs des quatre couleurs. La méthode `.split()` permet d'extraire chaque valeur et de lui affecter un nom mnémonique, tel que `red`. Comme nous disposons des valeurs, nous sommes prêts à mettre en œuvre la logique d'assignation des sons aux couleurs. Cette assignation est basée sur une expérimentation assez large avec des feuilles de papier colorées jusqu'à l'obtention d'un réglage correct. Nous avons utilisé des papiers avec une teinte claire, une teinte neutre et une teinte foncée pour chacune des trois couleurs de base pour étalonner le capteur. Nous avons mesuré à plusieurs reprises les valeurs de couleur respectives à une distance fixe et nous avons utilisé ces informations pour préciser les constantes dans le code Python. Par exemple, la commande *tone_red* est envoyée au Nano si la composante rouge, normalisée par rapport à l'intensité, est supérieure à 0,6. Ensuite l'intensité du vert est testée et la commande *tone_green* est envoyée si le seuil est dépassé. Enfin, la valeur de la composante bleue est testée. Si aucun seuil n'est atteint, « *no_color* » est envoyé au Nano, le faisant émettre la tonalité correspondante. Vous devrez probablement effectuer quelques expérimentations pour déterminer les constantes et adapter le système à votre garde-robe.

Version autonome

La version autonome du code est disponible sur la page web de cet article [7]. Elle respecte assez fidèlement l'exemple ci-dessus, sauf qu'au lieu d'attendre l'arrivée d'une commande sur la ligne série, le Nano attend à présent que l'on presse sur le bouton avant de lire le capteur et d'émettre le son adapté. De plus, l'information sur les couleurs est directement traitée dans le Nano, comme le montre l'extrait de code suivant :

```
if (red/clea > 0.6) {
  tone(buzz, 262, d);
} else if (green/clea > 0.3) {
...
```

Comme il est préférable d'éclairer systématiquement les vêtements avec la LED intégrée à la carte de liaison, il est possible de l'allumer avec `digitalWrite(led,HIGH)` juste avant la lecture des registres du TCS34725 et de l'éteindre juste après.
Enfin, il est possible de rendre le système portatif en utilisant un simple ATmega328, flashé avec un chargeur d'amorçage Arduino et relié au capteur. Alimenté par une batterie tant que l'on presse sur le bouton, le système se met en route et convertit continuellement la couleur en son jusqu'à ce que le bouton soit relâché afin de déconnecter la batterie. Le logiciel est également disponible sur la page web de l'article. Nous laissons la

À propos des auteurs

L'intérêt de Volker Ziemann pour l'électronique prend ses racines dans *Elektor* à l'ère de l'amplificateur Edwin 40W (milieu des années 70), mais il a en parallèle depuis cette époque étudié la physique et travaillé dans le domaine des accélérateurs de particules (SLAC aux États-Unis, CERN à Genève et maintenant à Uppsala en Suède). Étant donné que l'électronique joue un rôle important dans le contrôle et l'acquisition de données, son centre d'intérêt de toujours lui a été très utile au cours de sa carrière. Il enseigne à présent à l'université d'Uppsala et il est l'auteur de trois livres, l'un portant sur l'acquisition de données avec les cartes Arduino et le Raspberry Pi.
Annika Schlechter étudie afin d'obtenir un master en physique à l'université d'Heidelberg en Allemagne. Elle effectue actuellement un stage à l'Institut de recherche sur le cancer d'Heidelberg dans le cadre de sa licence. Le projet est né lors d'un échange Erasmus à l'université d'Uppsala en Suède au cours duquel Annika a participé au cours de Volker intitulé « Sensor to Report ».

Listing 1 : script Python pour l'ordinateur portable communiquant avec l'Arduino Nano.

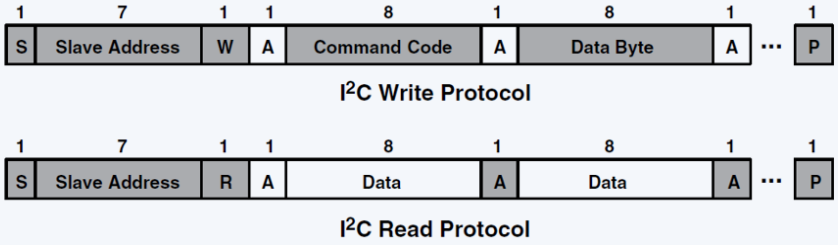
```
# colortosound_arduino.py
import serial, time

ser=serial.Serial("/dev/ttyUSB0",9600,timeout=1)
time.sleep(3);

ser.write(b"color?\n")  # write command to get colors
time.sleep(2);

reply=ser.readlines()   # read answer

# process answer to get color values
colors = reply[0].split()
for i, entry in enumerate(colors):
    colors[i]= int(entry.decode('utf-8').replace('\r\n', ''))

intensity = colors [0]; red = colors [1]
green = colors [2]; blue = colors [3]

# write command to control buzzer according to color

if red/intensity > 0.6:
    ser.write(b"tone_red\n")
elif green/intensity > 0.3:
    ser.write(b"tone_green\n")
elif blue/intensity > 0.24:
    ser.write(b"tone_blue\n")
else:
    ser.write(b"no_color\n")

ser.close()
```

Communication I²C

La communication entre deux circuits intégrés (I²C) est basée sur un protocole série synchrone et requiert deux fils. La communication suit une règle simple : détecter le niveau de tension dans l'un des fils, appelé SDA, après que la tension sur l'autre fil, appelé SCL, a été modifiée. Une lecture de ce type fournit un seul bit, huit bits consécutifs constituant un octet. Un des éléments du bus I²C, habituellement un microcontrôleur, orchestre la communication et fournit toujours le signal d'horloge. Les autres éléments ont la possibilité de participer à la communication si on s'adresse à eux et qu'on leur donne l'autorisation d'envoyer leurs données sur la ligne SDA. De plus, après chaque octet, on laisse au capteur un cycle d'horloge pour confirmer la réception de l'octet précédent.

Pour lancer la communication avec un capteur, le microcontrôleur envoie un octet. Les sept premiers bits de cet octet contiennent l'adresse tandis que le dernier bit indique si le contrôleur ou le capteur a le contrôle de la ligne SDA pour transmettre l'octet suivant. Si le contrôleur souhaite continuer l'envoi, par exemple pour modifier un registre dans le capteur, il envoie deux octets supplémentaires, l'adresse du registre et la nouvelle valeur. Si, par contre, le contrôleur veut lire un registre du capteur, il transmet le bit pour indiquer que le capteur a la main sur la ligne SDA ; le contrôleur continue alors à délivrer le signal d'horloge et lit la ligne SDA au rythme du signal d'horloge.

L'image du haut de la figure illustre l'écriture dans le capteur. Les bits qui sont grisés sont contrôlés par le microcontrôleur tandis que les bits qui ne sont pas grisés le sont par le capteur. Pour lancer une transaction, le microcontrôleur crée une condition de démarrage, désignée par « S », suivie des sept bits de l'adresse du capteur et du huitième bit activé pour indiquer que le contrôleur veut poursuivre l'envoi. Un bit « A » est retourné par le capteur pour indiquer qu'il a compris. Le second octet contient l'adresse du registre sur le capteur et un octet avec la valeur à inscrire dans le registre. Remarquez bien le bit noté « A » avec lequel le capteur confirme la réception de chaque octet. La lecture du capteur, illustrée dans l'image du bas, fonctionne à peu près de la même façon, simplement les huit bits du premier octet indiquent à présent que le capteur est autorisé à « parler » et que la ligne SDA est alors contrôlée par le capteur, tandis que le microcontrôleur doit confirmer chaque octet reçu.



Source : fiche technique TCS34725 / ams [6].

construction de cette version à nos lecteurs enthousiastes.
Pendant le cours « Sensor to Report », le capteur de couleurs a suscité l'intérêt. Une étudiante qui souhaitait surveiller la couleur du ciel sur une période prolongée afin d'analyser ensuite les données, a utilisé pratiquement la même configuration avec le Nano communiquant avec l'ordinateur hôte par l'intermédiaire de la ligne série. Elle a utilisé un script en Python sur un Raspberry Pi pour stocker les données dans une base de données MySQL, également hébergée sur le Raspberry Pi. Elle a ensuite utilisé Octave pour préparer les courbes des couleurs enregistrées pendant environ une semaine. Les courbes ont été enregistrées juste avant Noël et cela nous a tristement rappelé à quel point les jours peuvent être courts à Uppsala. D'autres projets d'acquisition de données dans le même esprit sont expliqués sur la page [8].

210051-04

Contributeurs

Conception et texte : **Annika Schlechter**, **Volker Ziemann**
Rédaction : **Jens Nickel**
Mise en page : **Sylvia Sopamena**

Des questions, des commentaires ?

Contactez Elektor (redaction@elektor.fr).

LIENS

[1] Carte de liaison TCS34725 : https://www.adafruit.com/product/1334
[2] Cours « Sensor to Report » : https://ziemann.web.cern.ch/ziemann/teaching/s2r19/
[3] Site web Arduino : https://www.arduino.cc/
[4] Tutoriels Arduino : https://www.arduino.cc/en/Tutorial/HomePage
[5] Site web Python : https://www.python.org/
[6] Fiche technique du capteur de couleurs TCS34725 : https://ams.com/tcs34725#tab/documents
[7] Téléchargement du logiciel : http://www.elektormagazine.fr/210051-04
[8] V. Ziemann, « A Hands-on Course on Sensors using the Arduino and Raspberry Pi », CRC Press, 2018.