La communication série avec Arduino

La communication série est indispensable pour dialoguer avec votre Arduino puisque c'est le mode de transmission utilisé pour communiquer avec la carte. Dans ce tuto je vais vous expliquer comment ça fonctionne avec des exemples pratiques pour faire communiquer une carte Arduino (le modèle n'importe pas) et votre PC grâce à une connexion USB. Nous utiliserons dans un premier temps l'IDE Arduino puis du code Python. Je suppose que vous avez quelques bases en programmation. Les exemples seront suffisamment simples pour que la connaissance préalable de python ne soit pas nécessaire.

Le tuto peut se réaliser indifféremment sous Windows, GNU/Linux, Mac ou autre. Néanmoins, les exemples fonctionnent tel quel uniquement sous linux. Sous d'autres systèmes, vous pouvez être amené à changer les chemins des ports. Le reste devrait rester inchangé.

Sommaire

Présentation

En communication série, on découpe l'information à transmettre en petits blocs de taille fixe avant de la transmettre. La taille des blocs correspond au nombre des lignes disponibles pour la transmission des données.

Ce type de communication s'oppose à la communication parallèle. En communication parallèle, il y a une ligne par bits à transmettre. Tous les bits sont donc transmis en même temps. Pour une même fréquence de communication, la communication parallèle est donc plus rapide.

L'avantage de la communication série sur la communication parallèle est qu'elle nécessite moins de lignes, donc moins de broches, donc moins de composants. Son coût est donc plus faible.

Les protocoles de communication série les plus connus sont :

- Le protocole USB
- Le protocole I2C
- Le protocole PCI Express

Pré-requis

Logiciels et bibliothèques à installer

- Arduino IDE : l'IDE officiel arduino. Le moyen le plus simple d'écrire et de télécharger les programmes sur la carte. Il est disponible sur le site officiel d'Arduino : http://arduino.cc/en/Main/Software#toc1
- Python3 : indispensable pour exécuter les certains programmes d'exemple (uniquement pour la partie communication entre un programme et la carte).
- Pyserial : c'est la bibliothèque python qui permet de faire de la communication série. Elle est normalement dans les dépôts de votre distribution linux. Elle peut aussi être installée via pip : `pip install pyserial`.

Vocabulaire

Commençons par définir quelques termes de vocabulaire avec lesquels vous pouvez ne pas être familier et que nous allons rencontrer tout au long de ce tuto :

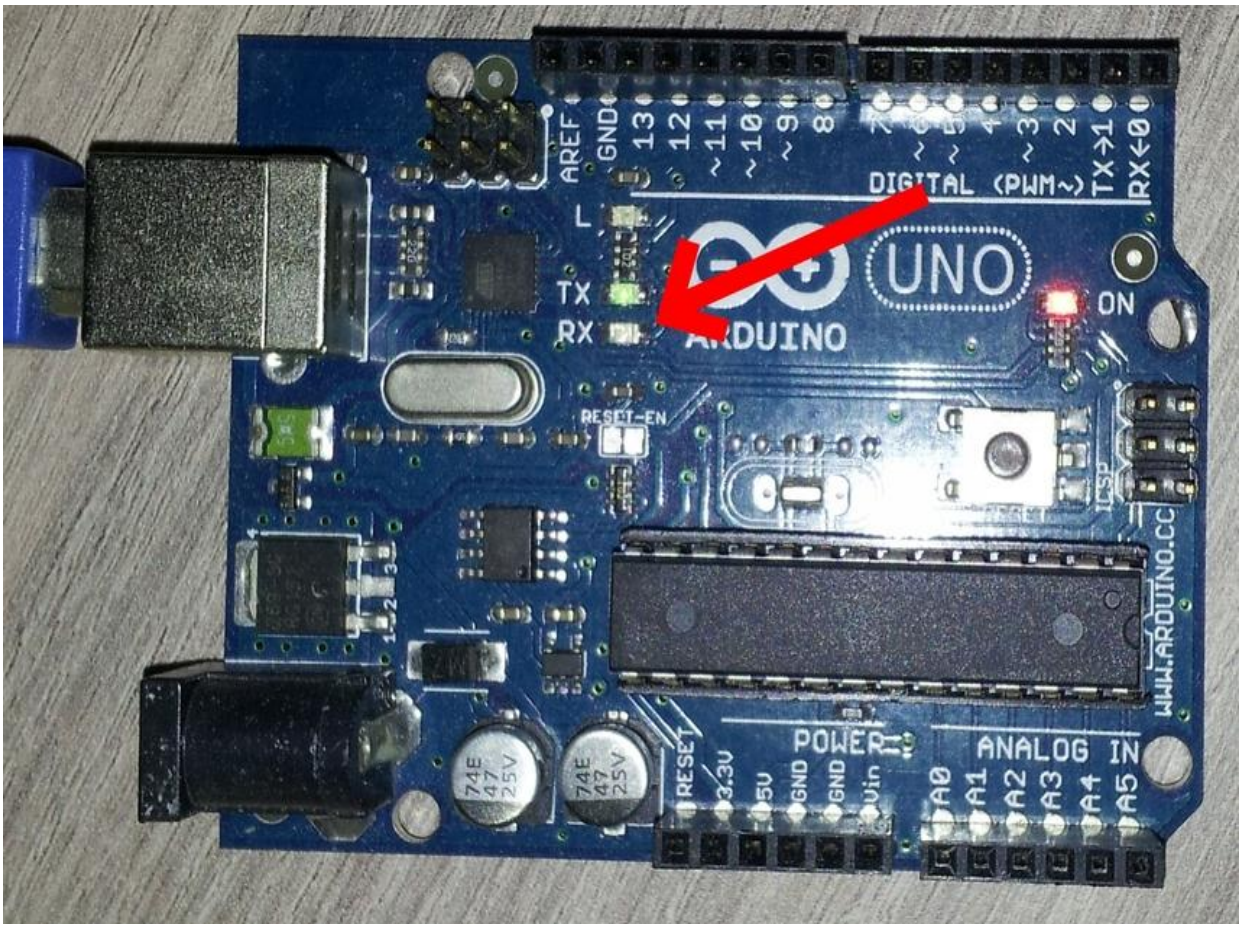
- Ascii : c'est l'acronyme de *American Standard Code for Information Exchange*, soit code américain pour l'échange d'information. Ce code permet de représenter avec des nombres compris entre 0 et 127 les caractères utilisés dans l'alphabet américain soit :
    - Les lettres, minuscules et majuscules
    - Les symboles de ponctuation : `. , ;`
    - Des caractères spéciaux : `$ # * +`
    - Pas les accents

- Baud (Bd) : Unité de mesure du nombre de symboles transmissibles par seconde. Il ne faut pas le confondre avec le nombre de bits par seconde (bps). Par exemple, considérons que l'on souhaite transmettre le symbole ascii `a` en 1 seconde exactement. D'après la table ascii `a` s'écrit en binaire `01100001`. Pour transmettre ce symbole en 1 seconde, il faut communiquer avec une vitesse de 1 Bd mais de 8 bps.
- Bytes (en python3) : Petite précision de vocabulaire spécifique à python 3. C'est un type de base du langage qui représente une chaîne d'octets. C'est ce type de donnée que nous manipulerons dans communication entre un programme et la carte

Communiquer avec le moniteur série

Avant de rentrer vraiment dans la communication, je vous signale que deux diodes présentes sur la arduino, peuvent être utiles :

- TX : s'allume lors d'une transmission
- RX : s'allume lors d'une réception



Communication la plus simple possible

Nous allons aborder notre premier exemple. Il consiste à envoyer des données sur le port série de la arduino de d'utiliser l'outil intégré à l'IDE pour voir les données envoyées. Lancez l'éditeur arduino et entrez le code ci-dessous :

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.println("Coucou");
}
```

Détaillons-le :

- Dans la fonction `setup`, on initialise simplement le port série en donnant sa vitesse avec `Serial.begin(9600)`. En effet, pour que la carte et l'ordinateur communique correctement, ils doivent « parler » à la même vitesse. Une vitesse différente est une erreur fréquente au début. Pensez à la vérifier si vous rencontrez des problèmes. On choisit ici 9600 Bd ce qui est une valeur classique. Nous verrons plus loin l'influence du nombre de bauds.
- Dans la fonction `loop`, on se contente d'envoyer `"Coucou"` avec `Serial.println("Coucou")`.

Vérifiez dans *Outils* > *Carte* que la bonne carte est sélectionnée et dans *Outils* > *Port série* que le bon port est choisi. Si tout est bon, utilisez le bouton *Téléchargements* pour envoyer le code sur la carte.
Ouvrez le moniteur série avec *Outils* > *Moniteur série*. Vérifiez en bas à droite que la valeur est bien à 9600 Bd. Si tout va bien, vous devriez voir le moniteur se remplir de `Coucou`.

Si ce n'est pas le cas, reprenez les étapes précédentes, vous avez raté quelque chose. Vous pouvez aussi regarder du côté des quelques problèmes fréquents.

Echo

Après cet exemple d'autant moins intéressant qu'il ne fonctionne que dans un sens, faisons un programme d'écho : la arduino va nous renvoyer exactement ce qu'on lui envoie. Si elle ne reçoit rien, elle envoie `Rien`. Voilà le code de la arduino :

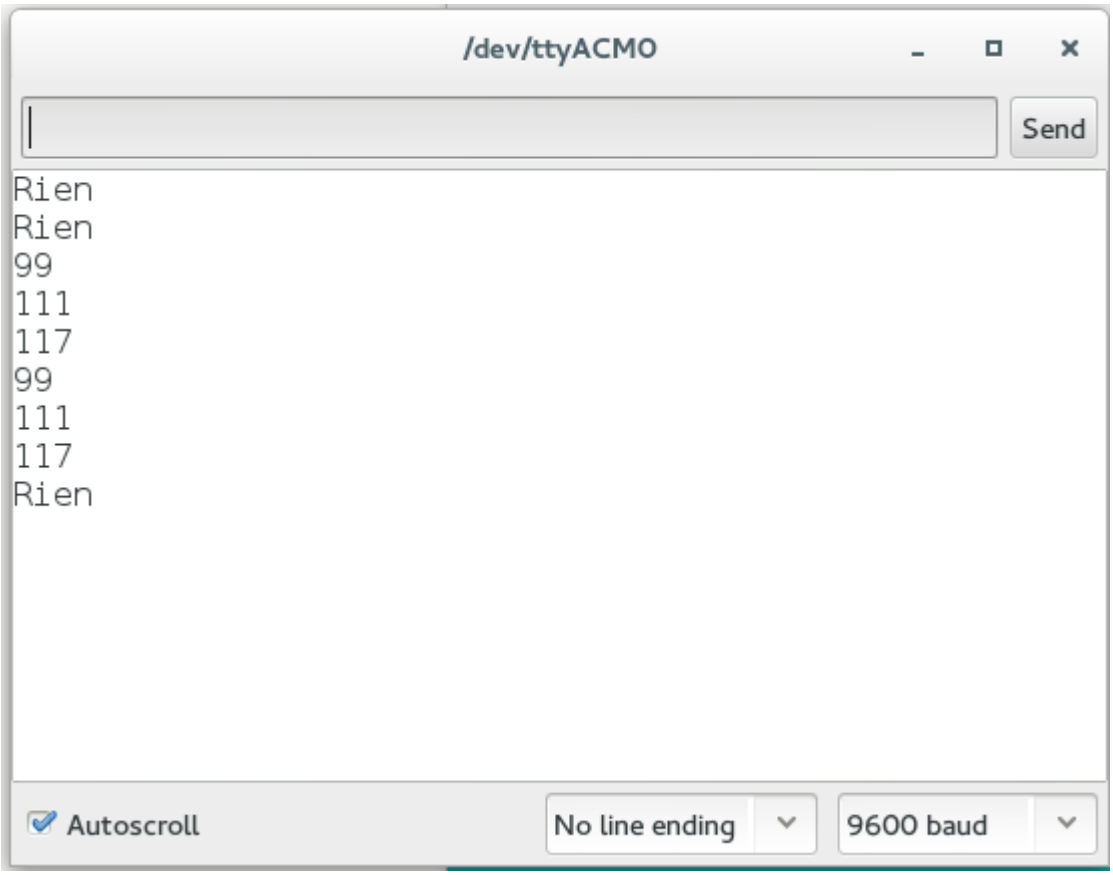
```
void setup() {
  Serial.begin(9600);
}

void loop() {
  if ( Serial.available() ) {
    int lu = Serial.read();
    Serial.println(lu);
  } else {
    Serial.println("Rien");
  }
  delay(2000);
}
```

Regardons-le d'un peu plus près. Pas de changement du côté de `setup`. Dans `loop` :

- Nous commençons par vérifier si des données en attente de lecture sur le port avec : `Serial.available()`. Cette fonction renvoie `true` si telle est le cas.
- Si on a des données, on définit une variable de type `int` dans laquelle on place une donnée en attente. Puis on affiche cette donnée.
- Sinon, on affiche `Rien`.
- Enfin, on attend 2 secondes histoire d'avoir le temps de voir ce qui se passe.

Testez ce code. Pour envoyer une donnée à la carte, tapez du texte dans la barre du haut puis presser *Entrée* ou appuyez sur *Envoyer*. Vous devriez obtenir quelque chose comme ça (j'ai tapé *coucou*) :



Mais ce n'est pas ce que j'ai tapé ! En effet, `Serial.read()` lit un octet de données sur le port série. Ni plus, ni moins. Et c'est la seule manière de lire les données. Mais rassurez-vous, vous pouvez facilement donner un sens à ce qui s'affiche : les données sont encodées en ascii. Par conséquent, le numéro qui s'affiche est la version décimale du code ascii. Vous pouvez consulter la table pour convertir ce code dans le caractère standard.

Lire des entiers

Dans cette partie, nous allons utiliser la table ascii pour convertir le nombre reçu en quelque chose d'exploitable. Nous ferons tout d'abord un programme qui lit et affiche correctement un nombre puis un second programme qui fera des opérations sur deux entiers que vous choisirez.

Lecture basique d'un entier

Ce n'est pas très compliqué. Il suffit de prendre la table ascii et de constater que les chiffres sont codés en décimal entre 48 et 57 puis de décaler.

```
int byte_read = 0; ///< The current byte
int recieved_integer = 0; ///< The current recieved integer
```

```
void setup() {
  Serial.begin(9600);
}

boolean is_a_number(int n)
{
  return n >= 48 && n <= 57;
}

int ascii2int(int n, int byte_read)
{
  return n*10 + (byte_read - 48);
}

void loop() {
  recieved_integer = 0;
  while ( Serial.available() ) {
    byte_read = Serial.read();
    if ( is_a_number(byte_read) ) {
          recieved_integer = ascii2int(recieved_integer,
byte_read);
    }
  }
  Serial.println( recieved_integer );
  delay(1000);
}
```

Détaillons ce programme :

- On définit les variables globales en début de programme. Ça ne sert à rien de les redéfinir à chaque tour de boucle.
- Ensuite on définit deux fonctions :
  - `boolean is_a_number(int n)` qui renvoie vrai si on lui donne un nombre ascii codé en décimal.
  - `int ascii2int(int n, int byte_read)` qui renvoie le nombre sous la forme d'un entier. Elle prend en paramètre l'entier déjà calculé ainsi que l'octet à ajouter.
- Dans `loop` : on fait la conversion si un entier est disponible.

Je vous laisse constater que ça marche bien.

Faire des maths avec deux entiers

Maintenant on va faire un peu mieux : lire deux entiers séparés par une virgule et les additionner. Le code est ci-dessous :

```
const int baudrate = 9600;
```

```
int byte_read = 0; ///< The current byte read.
int coords[2]; ///< Contains x and y.
int index = 0; ///< 0: reading x, 1: reading y.
int separator = 44; ///< The separator between the integers
(44: ,)

void setup()
{
    Serial.begin(baudrate);
}


boolean is_a_number(int n)
{
  return n >= 48 && n <= 57;
}


int ascii2int(int n, int byte_read)
{
  return n*10 + (byte_read - 48);
}


void loop()
{
  coords[0] = 0;
  coords[1] = 0;
  index = 0;
  while ( Serial.available() > 0 )
    {
      byte_read = Serial.read();
      if ( is_a_number(byte_read) )
        {
          coords[index] = ascii2int( coords[index],
byte_read );
        }
      else if ( byte_read == separator )
        {
          ++index;
        }
    }

  if ( index )
    {
      Serial.print("x + y = ");
      Serial.println(coords[0] + coords[1]);

      Serial.print("x * y = ");
      Serial.println(coords[0] * coords[1]);
```

```
    }
  else
    {
      Serial.println();
    }
  delay(1000);
}
```

Décortiquons :

- Concernant les variables :
  - Nos deux nombres seront stockés dans un tableau d'entier.
  - `index` nous permet de savoir si on lit x ou y.
- Dans `loop` :
  - On prend soin de remettre à 0 les variables pour éviter les problèmes.
  - Puis suivant si l'octet lut représente un nombre ou le séparateur, on lit un nombre ou on incrémente l'index.
  - Concernant les calculs : on vérifie que `index` n'est pas à 0 et donc qu'on a bien lut deux nombres. Si tel est le cas, on fait l'addition et la multiplication de x et y et on affiche le résultat. La fonction `print` affiche ses paramètres sans revenir à la ligne comme `println`.

Vous pouvez vous amuser à écrire une calculatrice plus complète si vous voulez. Vous pouvez même envoyer l'opération à effectuer si vous voulez. Utilisez éventuellement l'instruction switch.

Communication entre un programme et la carte

Cette partie reprend en partie ce qui a déjà été évoqué ici mais de façon plus détaillée et un peu moins complète.

Lecture

Nous allons maintenant faire communiquer un programme de votre ordinateur (autre que l'IDE Arduino) et la carte. Le code sera écrit directement dans l'interpréteur, mais vous pouvez sans problème le mettre dans un fichier puis l'exécuter. Il sera écrit en python, mais les principes restent les mêmes dans les autres langages. Reprenons le code « echo » pour la carte :

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  if ( Serial.available() ) {
    int lu = Serial.read();
```

```
    Serial.println(lu);
  } else {
    Serial.println("Rien");
  }
  delay(2000);
}
```

Lancez l'interpréteur python sur votre PC et importer la bibliothèque pyserial :

```
Python 3.3.2 (default, Mar  5 2014, 08:21:05)
[GCC 4.8.2 20131212 (Red Hat 4.8.2-7)] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> from serial import Serial
```

Créez la communication série avec (en remplaçant `/dev/ttyACM0` par le nom de votre port) :

```
>>> serial_port = Serial(port='/dev/ttyACM0', baudrate=9600)
```

Pour lire des données, rien de plus simple :

```
>>> serial_port.read()
b'R'
>>> serial_port.readline()
b'Rien\r\n'
```

Les résultats sont de type `bytes`. Si vous comptez faire des manipulations par la suite (hors transmission série), je vous conseille de passer dans le type string plus adapté (ou un autre type qui représente correctement vos données) :

```
>>> lu = serial_port.readline()
>>> lu
b'Rien\r\n'
>>> type(lu)
<class 'bytes'>
>>> chaine = lu.decode('ascii')
>>> chaine
'Rien\r\n'
>>> type(chaine)
<class 'str'>
```

Écriture

Pour écrire des données c'est aussi très simple :

```
>>> while True:
...     nombre = input('Entrez un chiffre : ')
...     serial_port.write(nombre.encode('ascii'))
...     serial_port.readline()
...
Entrez un nombre : 78
1
b'55\r\n'
```

Le nombre `1` est la valeur de retour de `serial_port.write` qui renvoie la longueur de la chaîne écrite.

Influence du nombre de bauds

Comme vous avez dû le comprendre en lisant vocabulaire, plus le nombre de bauds est élevé, plus on peut transmettre de données dans un temps court. Nous allons illustrer ceci dans cette partie. Télécharger tout d'abord le fichierdonnee.txt qui contient le texte à envoyer. Enregistrez de façon à pouvoir l'ouvrir facilement avec l'interpréteur python. Programmez votre arduino avec le code suivant :

```
void setup() {
  Serial.begin(117500);
}

void loop() {
  while ( Serial.available() ) {
    Serial.read();
  }
}
```

Puis dans l'interpréteur python, tapez :

```
>>> port = Serial(port='/dev/ttyACM0', baudrate=117500)
>>> f = open('donnees.txt', 'r')
>>> data = f.read()
>>> a.write(data.encode('ascii'))
```

L'interpréteur devrait répondre rapidement `8000`. Remplacez maintenant `117500` par `4800` dans les deux codes. C'est **beaucoup** plus long, non ?

Le principal est d'utiliser la bonne vitesse de transmission pour son application. Avec une grande vitesse, vous transmettez plus d'informations mais cela peut poser des

problèmes sur les grandes distances. À vous de tester et de choisir la bonne, sachant qu'en général sur des cas simples, 9600 Bd est suffisant.

Conclusion

À partir de ces informations, vous devriez pouvoir faire ce que vous voulez tant que ce n'est pas trop compliqué. Je publierais sûrement d'ici quelques temps un tuto plus complet sur pyserial. En attendant, vous pouvez aller consulter la documentation.

En résumé

- Choisir la bonne vitesse de communication.
- Que les deux programmes qui communiquent le fasse à la même vitesse.
- Se souvenir que l'arduino lit les données en code ascii.
- Ne pas désespérer si ça ne fonctionne pas.

Quelques problèmes fréquents

Problème de téléchargement

Ils sont malheureusement courants et je n'ai pas de méthode miracle pour les régler. Vous pouvez :

- Vérifiez le port de communication
- Débranchez/Re-branchez la carte
- Relancez l'éditeur

Lecture de données de taille indéterminées

Lors de tous nos tests de lecture, nous avons utilisé soit des int soit des tableaux de taille fixe de int. Vous ne pouvez pas lire des données de taille indéterminée avec la arduino. Par exemple, faire :

```
String data = Serial.read();
```

En effet, contrairement à votre ordinateur qui a *beaucoup* de mémoire, une arduino est beaucoup plus limitée : pas plus de quelques kilos octets. La lecture de données sous cette forme empêche le compilateur de connaître la taille nécessaire pour la donnée et va donc refuser de compiler le code.
Dans tous les cas, souvenez-vous que la mémoire est *très* limitée et que vous ne pourrez donc pas tout traiter.