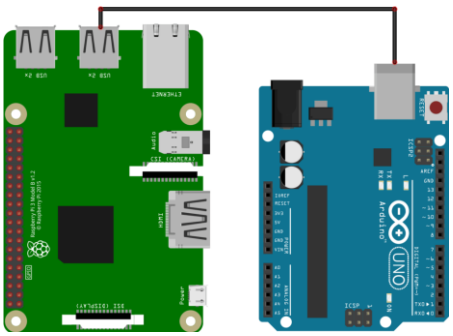
Fiche Tutoriel 14

Arduino - Raspberry : communication et data transfert par le port USB

Généralités :

Dans certains projets, il peut être intéressant d'établir une communication série entre Raspberry Pi et Arduino. Par exemple, vouloir transférer les données de capteurs analogiques contrôlés par l'Arduino (non pilotable par la Raspberry) sur une rapsberry qui servira de serveur ou tout simplement de relais pour envoyer ces données sur un site. Il est ainsi possible de coupler la puissance de calcul et les interface sans fil du Raspberry Pi avec les entrées-sorties et la collection de modules Arduino. Lorsque vous travaillez avec des appareils électroniques, la communication est essentielle. C'est l'une des choses les plus importantes car elle de passer d'une application très basique à des applications plus complexes. L'une des solutions pour faire communiquer une Arduino et une Raspberry et d'utiliser le port USB. Cette communication série est simplement un moyen de transférer des données. Les données seront envoyées séquentiellement, un peu à la fois (1 octet = 8 bits).



Branchement entre Raspberry et Arduino par le port USB

Protocole UART :

Plus précisément, lorsque vous utilisez Serial avec Arduino et Raspberry Pi, vous utilisez le protocole UART. UART signifie « Universal Asynchronous Reception and Transmission». Fondamentalement, il s'agit d'un protocole multi-maître asynchrone basé sur la communication série, qui vous permettra de communiquer. Soyez rassuré, il y a des bibliothèques qui vont gérer tout cela.

Multi-master signifie que tous les appareils connectés seront libres d'envoyer des données quand ils le souhaitent. C'est l'une des principales différences avec les protocoles maître-esclaves, où seul le dispositif principal peut initier une communication. Habituellement, vous utilisez d'autres protocoles tels que I2C lorsque vous avez besoin de configurations maître-esclaves.

L'Arduino a un UART que vous pouvez utiliser soit avec un câble USB ou à partir des broches RX / TX (ne l'utilisez pas avec les deux en même temps). Pour les connexions :

- Tx GPIO14(RPI) <-> Rx 0(Arduino)
- Rx GPIO15(RPI) <-> Tx 1(Arduino)
- GND (RPI) <-> GND(Arduino)

Sur le Raspberry Pi, vous pouvez connecter de nombreux périphériques Serial sur les ports USB. Chacun aura un nom d'appareil différent (nous verrons comment les trouver plus tard dans ce tutoriel).

Coté Arduino :

Sans le savoir vous avez déjà utilisé la communication série sous Arduino. En effet quand vous imprimé vos donnes dans la console Arduino vous avez employé Serial.begin et Serial.println(« texte ou valeur »). C'est exactement la même chose que nous allons faire dans la communication série sauf que le port de l'Arduino enverra les données sur la Raspberry

Tapons le code suivant :

```
void setup() {
  // put your setup code here, to run once:
Serial.begin(9600); // on initialise le port serie à 9600 baud
}

void loop() {
  // put your main code here, to run repeatedly:
Serial.println("bonjour de l'arduino"); // on écrit sur le port serie la phrase
delay (10000);                          // on attend 10 secondes
}
```

nous initialisons la communication en série, et choisissons un taux de baud, de 9600. 9600 est un taux de baud couramment utilisé, et aussi un taux assez faible. Dans vos futures applications, vous pouvez utiliser un taux de baud de 57600 ou même 115200 sans aucun problème.

Coté Raspberry :

La les choses se compliquent un peu plus. Il faut tout d'abord déterminer sur quel port votre Arduino a été branché. Pour cela ouvrir la console de ligne de commande (cf fiche pratique 3) sans avoir branché l'arduino tapez la commande :

ls /dev/tty*

```
pi@raspberrypi:~ $ ls /dev/tty*
/dev/tty    /dev/tty19  /dev/tty3   /dev/tty40  /dev/tty51  /dev/tty62
/dev/tty0   /dev/tty2   /dev/tty30  /dev/tty41  /dev/tty52  /dev/tty63
/dev/tty1   /dev/tty20  /dev/tty31  /dev/tty42  /dev/tty53  /dev/tty7
/dev/tty10  /dev/tty21  /dev/tty32  /dev/tty43  /dev/tty54  /dev/tty8
/dev/tty11  /dev/tty22  /dev/tty33  /dev/tty44  /dev/tty55  /dev/tty9
/dev/tty12  /dev/tty23  /dev/tty34  /dev/tty45  /dev/tty56  /dev/ttyAMA0
/dev/tty13  /dev/tty24  /dev/tty35  /dev/tty46  /dev/tty57  /dev/ttyprintk
/dev/tty14  /dev/tty25  /dev/tty36  /dev/tty47  /dev/tty58
/dev/tty15  /dev/tty26  /dev/tty37  /dev/tty48  /dev/tty59
/dev/tty16  /dev/tty27  /dev/tty38  /dev/tty49  /dev/tty6
/dev/tty17  /dev/tty28  /dev/tty39  /dev/tty5   /dev/tty60
/dev/tty18  /dev/tty29  /dev/tty4   /dev/tty50  /dev/tty61
pi@raspberrypi:~ $
```

Brancher votre Arduino sur l'un des port USB et retaper la commande

```
pi@raspberrypi:~ $ ls /dev/tty*
/dev/tty    /dev/tty19  /dev/tty3   /dev/tty40  /dev/tty51  /dev/tty62
/dev/tty0   /dev/tty2   /dev/tty30  /dev/tty41  /dev/tty52  /dev/tty63
/dev/tty1   /dev/tty20  /dev/tty31  /dev/tty42  /dev/tty53  /dev/tty7
/dev/tty10  /dev/tty21  /dev/tty32  /dev/tty43  /dev/tty54  /dev/tty8
/dev/tty11  /dev/tty22  /dev/tty33  /dev/tty44  /dev/tty55  /dev/tty9
/dev/tty12  /dev/tty23  /dev/tty34  /dev/tty45  /dev/tty56  /dev/ttyACM0
/dev/tty13  /dev/tty24  /dev/tty35  /dev/tty46  /dev/tty57  /dev/ttyAMA0
/dev/tty14  /dev/tty25  /dev/tty36  /dev/tty47  /dev/tty58  /dev/ttyprintk
/dev/tty15  /dev/tty26  /dev/tty37  /dev/tty48  /dev/tty59
/dev/tty16  /dev/tty27  /dev/tty38  /dev/tty49  /dev/tty6
/dev/tty17  /dev/tty28  /dev/tty39  /dev/tty5   /dev/tty60
/dev/tty18  /dev/tty29  /dev/tty4   /dev/tty50  /dev/tty61
pi@raspberrypi:~ $
```

Votre Arduino est donc sur le port ttyACM0 (dans ce cas mais peut etre ACM1 ACM2…), attention cependant si vous débranché votre Arduino et que la rebranchée le numéro peut varier à vous de vérifier à chaque fois.

Quand votre port est identifié on peut passer à la programmation sous python pour lire les valeurs envoyées par l'Arduino.

Comme précédent il faut tout d'abord importer les librairies dont nous avons besoin. Le port série nécessite lui la bibliothèque serial

Une fois fait on créer un objet serial qui aura pour propriété l'adresse du port et la vitesse en baud que vous avez défini dans le programme Arduino. Puis ensuite on lis ce qui arrive sur le port série par la fonction (serial.read()). Il peut s'avérer utile de toujours démarrer la lecture en ayant au préalable nettoyé les entrées et sortie par les fonctions : serial.flushInput() et serial.flushOutput() Vous pouvez donc tapez le code suivant sous python :

```
#!/usr/bin/python3 #-*- coding: latin-1 -*-
import  serial # on importe la librairie serie
import  time   # on importe la librairie temps


serial=serial.Serial('/dev/ttyACM0',9600) #on defini un objet serial avec l'adresse du port et la vitesse


while True: #on effectue un boucle infine
    serial.flushInput();serial.flushOutput() # on nettoie les buffer
    number=(serial.read()) # on lit sur le port serie et on affecte dans une variable
    print(number) # on imprime dans la console
```

Shell

```
b'\n'
b'b'
b'n'
b'r'
b' '
b'a'
b'i'
b'\n'
```

Dans la console nous voyons effectivement une partie des données envoyées par l'arduino mais la phrase n'est pas complète. Tout simplement parce que la fonction serial.read() lit les données bytes à bytes et que le temps d'afficher dan la console la première lettre et de relire les données sur le port série, l'Arduino aura dejà envoyer la 4ème lettre. L'une des solutions et de compter le nombre de bytes a envoyer ici 20 caractère en comptant les espaces la dessus il faut rajouter deux caractère spéciaux que sont le / r et /n correspondant respectivement :

/r correspondant au retour chariot

/n correspondant au changement de ligne

Soit un total de 22 caractères. Cependant cette solution suppose de connaitre la longueur de la chaine de caractère envoyée, ce qui dans certains cas n'est pas toujours connu (envois de données de capteur par exemple). C'est pourquoi on preferera employer la fonction serial.readline() qui elle attendra le caractère /n pour savoir que la réception est fini

Et pour supprimer les caractère /r et /n on lui assigne la propriété, ainsi que le caractère b' (signifie que ce qui suit est un byte).

.decode('utf8', errors='replace')

UTF-8 est un encodage universel qui a pour objectif de réunir les caractères utilisés par toutes les langues. Il n'y a donc en théorie plus de problèmes de communication si tous les programmes sont encodés avec de l' **UTF8** . Le 'replace' remplacera tous les caractères Unicode non encodables par un point d'interrogation (?)

Le code devient donc :

```python
#!/usr/bin/python3 #-*- coding: latin-1 -*-
import  serial # on importe la librairie serie
import  time   # on importe la librairie temps


serial=serial.Serial('/dev/ttyACM0',9600) #on defini un objet serial avec l'adresse du port et la vitesse


while True: #on effectue un boucle infine
    serial.flushInput();serial.flushOutput() # on nettoie les buffer
    number=(serial.readline().decode("utf8", errors="replace")) # on lit sur le port serie et on affecte dans une variable
    print(number) # on imprime dans la console
```

```
hell
bonjour de l'arduino

bonjour de l'arduino
```

Vous pouvez de la même manière envoyer des valeurs décimales, des entiers, le résultats de calculs etc… puisque quand vous utilisez la fonction serial.println(‘data’) vous envoyer systématiquement une chaine de caractère. Sur la Raspberry si vous souhaitez utiliser les datas envoyées comme nombre il faudra les retransformer.