Fiche Tutoriel 15

Arduino - Raspberry : communication par le port I2C

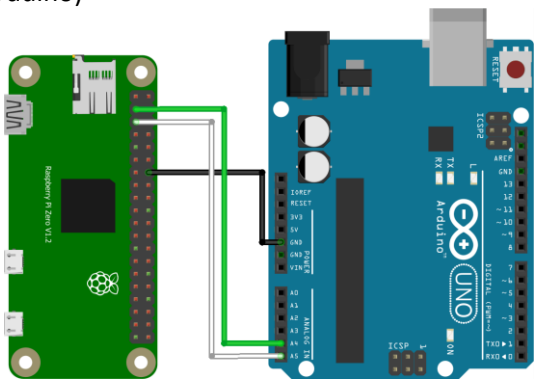
Généralités :

Par rapport à la communication série (cf fiche pratique 14), la communication par le port I2C est un bon choix de protocole de communication si la carte Raspberry Pi et la l' Arduino sont à proximités. De plus le bus ou port I2C permet de rajouter plusieurs appareils sur le même bus et de décupler les capacités du Raspberry. C'est-à-dire qu'a partir d'un port I2C de la Raspberry on peut piloter plusieurs Arduino, chose impossible par le port série (limité à 4 sur la Raspberry).

Par rapport à la communication série, les protocoles de transfert des données ne s'effectuera qu'avec des requêtes maître-esclaves, les données ne seront envoyées quand cela sera nécessaire.

Pour établir la communication I2C entre Raspberry Pi et Arduino, il nous faut relier physiquement le bus qui utilise 3 broches. Les broches utilisées par la communication I2C sont généralement SDA (Serial Data Line) sur laquelle sont envoyées les données et SLC (Serial Clock Line) qui est l'horloge de synchronisation. Les masses des deux cartes doivent être reliées pour établir une référence commune de potentiel.

- SDA BCM2(RPI) <-> SDA A4(Arduino)
- SCL BCM3(RPI) <-> SCL A5(Arduino)
- GND (RPI) <-> GND(Arduino)



Port I2C :

Le terme I2C signifie (inter-integrated circuits). C'est un protocole de communication développé par Philips Semiconductors pour le transfert de données entre un processeur central et plusieurs esclave (capteurs, encodeurs etc…) sur la même carte de circuit en utilisant seulement deux fils communs.

Cependant pour mettre en place la communication I2C certaines étapes sont nécessaires du coté Arduino et Raspberry. Nous ne verrons que l'exemple ou l'arduino est l'esclave et la rapsberry le maitre

Coté Arduino :

Il faudra pour l'Arduino utiliser la bibliothèque **wire.h** pour pouvoir initialiser la communication I2C puis donner une adresse à la carte Arduino .

Comme l'Arduino est l'esclave il faudra définir une fonction associée à la commande **wire.onRequest(fonction)**, qui lors de la requête du maitre (Raspberry) effectuera la fonction demandé. Dans cette fonction on enverra les données par la fonction **Wire.write('données')**.

Cependant attention la fonction Wire.wire ne peut envoyer que trois types de variables :
Wire.write(valeur) : valeur à envoyer comme simple byte (donc un entier et pas de décimale)
Wire.write(string) : chaine de caractère comme une série de bytes
Wire.write(data, length) : un tableau de données sous formes de plusieurs bytes(32 bits pour chaque), et length déterminant le nombre de bytes à transmettre.

Il est à noter que du côté de la Raspberry les données ne pourront être reçues que bytes à bytes Donc pour un entier seul on enverra une seule valeur (premier cas), mais dans le cas des plusieurs valeurs à envoyer, ou des chiffres décimaux on transformera les valeurs en chaine de caractère ou en tableaux (essentiellement).
Voici le code Arduino que vous pouvez taper :

```
#include <Wire.h>           // on charge la bibliothèque I2C
#define SLAVE_ADDRESS 0x04 //on definit l'adresse de l'esclave
 char value [14];          // on créer une variable tableau de chaine de caractère de 14 place
 float temperature=32.56; // on definit une température en décimale
 float humidity=92.5 ;     //on definit une humidité en décimale

void setup() {
  // On initialise la communication I2C avec l'adresse de l'esclave
   Wire.begin(SLAVE_ADDRESS); // an address makes this unit a Slave
  // on defini la fonction appelé lors de la requete du maitre
    Wire.onRequest(sendData);
}

void loop() {
  delay(1000); // on attend 1 seconde
}

void sendData(){
  String temp=String(temperature,2); // on transforme le chiffre decimale en caractère
  String humid=String(humidity,2);   // on transforme le chiffre decimale en caractère
//on definit une variable de caractère pour concaténer les deux variables temp et humid séparé par ;
String DataTotal=(temp+";"+humid+";");
// on envoie la string dans le tableau d'ou la nécessité d'avoir assez de cases
DataTotal.toCharArray(value,14);
     Wire.write(value); // on evoie les données par le port I2C
  delay(250);  // on attend 250 milisecondes
}
```

Coté Rapsberry :

Avant de pouvoir commencer le script python il faut absolument activer le port I2C de la Raspberry comme décrit sans la fiche pratique 6. Comme nous l'avons vu nous avons déclaré une adresse pour esclave. Pour savoir si cette adresse est reconnue par la Raspberry, il vous suffit de brancher l'Arduino en I2C sur la Raspberry et de l'alimenter. Sous Raspbian ouvrir la console de ligne de commande (cf fiche pratique 3) et tapez : sudo i2cdetect -y 1. Vous devriez avoir ce genre de réponse

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- 04 -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
pi@raspberrypi:~ $ Daffy,
bash: Daffy, : commande introuvable
pi@raspberrypi:~ $
```

La connection I2C est sur l'adresse 4

Maintenant que toutes les vérifications sont faites, nous pouvons passer au code python. Comme tout script python on va d'abord importer la bibliothèque permettant de gérer la communication I2C. Cette dernière est appelée **smbus**. Il faudra ensuite déclarer quelle est l'adresse de esclave a contacter puis de créer un objet gérant le tout par **: bus=smbus.SMbus(1).** Le chiffre (1) est valable en fait pour les raspberrys model 3&4 pour les versions antérieures il faudra remplacer le (1) par (0).

C'est tout pour les variables. Maintenant pour lire sur le port I2C les données transmissent par l'Arduino, et les stocker dans une variable l'on peut appliquer la fonction suivante : **data= bus.read_i2c_byte_data(slave_adresse, registre).** Cependant cette fonction pose un problème elle ne lit qu'un seul byte des données transmises et le registre correspond à la position du byte envoyé. Si vous ne connaissait absolument pas la taille des variables envoyées et que vous ne pouvez joindre les données ensemble cela posera de gros problèmes.

La solution est d'employer une autre fonction : **data=bus.read_i2c_block_data(slave_adresse, registre, taille).** Cette fonction permettra de lire sur le bus I2C un bloc de data transmises en une seule fois dans une variable de type liste**.** Pour les paramètres il faudra donc mettre l'adresse de l'esclave, la byte de départ et la longueur des bytes que l'on souhaite lire. Généralement les données transmises et lues de cette manière sont en générales une liste ou un tableau (attention la byte de départ commence à 0). En fin on affichera les données reçues dans la console.

<u>Code commenté :</u>

```python
#!/usr/bin/python3

import smbus as smbus # on importe la bibliothèque pour le gestion I2C
import time # on importe la bibliothèque temps

addr = 0x04 # on defini l'adresse de l'esclave
bus=smbus.SMBus(1) # on créé un objet I2C bus


while True : # on lance une boucle infinie
    #on utilise la fonction pour lire un bloc de data sur 32 bytes du port I2C
    #avec l'adresse de l'esclave le byte de départ et le nombre de bytes désiré
    block=bus.read_i2c_block_data(addr,0,14)
    print(block) # on imprime la variable recue
    time.sleep (15) # on attend 15 secondes
```

Shell

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run 'I2C test.py'

 [51, 50, 46, 53, 54, 59, 57, 50, 46, 53, 48, 59, 255, 255]
 [51, 50, 46, 53, 54, 59, 57, 50, 46, 53, 48, 59, 255, 255]
 [51, 50, 46, 53, 54, 59, 57, 50, 46, 53, 48, 59, 255, 255]
 [51, 50, 46, 53, 54, 59, 57, 50, 46, 53, 48, 59, 255, 255]
 [51, 50, 46, 53, 54, 59, 57, 50, 46, 53, 48, 59, 255, 255]
 [51, 50, 46, 53, 54, 59, 57, 50, 46, 53, 48, 59, 255, 255]
 [51, 50, 46, 53, 54, 59, 57, 50, 46, 53, 48, 59, 255, 255]
```

On voit effectivement des données sont transmises, cependant nous ne retrouvons pas les données envoyées par l'Arduino et notamment les valeurs décimales de la température et de l'humidité. En fait le port I2C transmet les données sous forme ASCII.

51 , 50, 46, 53, 54, 59, 57, 50, 46, 53, 48, 59 en code ASCII correspond en alphabet universel (UTF8 cf fiche pratique 14) à 32.56 ;92.50 ;

Ce sont bien les variables déclarées dans l'Arduino Il faut donc maintenant mettre en forme toutes ces données

Mise en forme des données reçues par I2C :

Pour mettre en forme les données reçues dans python il faut considérer 2 choses :

-La première est que les données sont dans une liste c'est-à-dire que chaque chiffre est située dans une case de la liste

- La deuxième c'est pour avoir la donnée complète il faut joinde chaque case de la liste et la décoder en alphabet

On utilisera sous python la fonction :

Decode="".join(map(chr,data)). Cette fonction créer une variable decode vide au départ dans laquelle on va joindre tous les éléments de la liste (map) transformés en caractère par (chr,data). La variable générée sera une chaine de caractère (String). Si l'on veut avoir maintenant une nouvelle liste dans laquelle chaque case contiendra une valeur (case 1 la température, case 2 l'humudité etc…) Il faut employer la fonction :

decode=decode.split(";"). Ou en fait on séparera la variable dès que l'on trouvera le caractère « ; ».

le code avec mise en forme des données :

```
1  #!/usr/bin/python3
2
3  import smbus as smbus # on importe la bibliothèque pour le gestion I2C
4  import time # on importe la bibliothèque temps
5
6  addr = 0x04 # on defini l'adresse de l'esclave
7  bus=smbus.SMBus(1) # on créé un objet I2C bus
8
9
10 while True : # on lance une boucle infinie
11     #on utilise la fonction pour lire un bloc de data sur 32 bytes du port I2C
12     #avec l'adresse de l'esclave le byte de départ et le nombre de bytes désiré
13     block=bus.read_i2c_block_data(addr,0,14)
14     # on joint tous les bytes du tableau en une seule variable n que l'on decode en caractère
15     n="".join(map(chr,block))
16     n=n.split(";") #on separe tous les caractères par le ; et on creer un tableau
17     print(n)# on imprime le tableau
18     time.sleep (15) # on attend 5 secondes
19
```

Shell

```
Python 3.7.3 (/usr/bin/python3)
>>> %Run 'I2C test.py'
  ['32.56', '92.50', 'ÿÿ']
```

Comme pour la communication série si vous souhaitez utiliser les datas envoyées comme nombre il faudra les retransformer.