Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

ÉDITION SPÉCIALE SÉRIE PROGRAMMATION

ÉDITION SPÉCIALE
SÉRIE PROGRAMMATION

PYTHON
Dans le Monde Réel

Volume Douze
Parties 67 à 72

Au sujet du Full Circle

Le Full Circle est un magazine gratuit, libre et indépendant, consacré à toutes les versions d'Ubuntu, qui fait partie des systèmes d'exploitation Linux. Chaque mois, nous publions des tutoriels, que nous espérons utiles, et des articles proposés par des lecteurs. Le Podcast, un complément du Full Circle, parle du magazine même, mais aussi de tout ce qui peut vous intéresser dans ce domaine.

Clause de non-responsabilité :

Cette édition spéciale vous est fournie sans aucune garantie ; les auteurs et le magazine Full Circle déclinent toute responsabilité pour des pertes ou dommages éventuels si des lecteurs choisissent d'en appliquer le contenu à leurs ordinateur et matériel ou à ceux des autres.

Spécial Full Circle Magazine

Full Circle

LE MAGAZINE INDÉPENDANT DE LA COMMUNAUTÉ UBUNTU LINUX

Bienvenue dans une nouvelle édition spéciale consacrée à un seul sujet !

En réponse aux requêtes des lecteurs, nous avons réuni le contenu de certains articles consacrés à la programmation en python.

Pour l'instant, il s'agit d'une réédition directe de la série **Programmer en Python**, **parties 67 à 72**, des numéros 110 à 115, par l'incomparable professeur en Python Greg Walters.

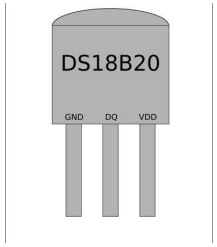
Veuillez considérer l'origine de la publication ; les versions actuelles du matériel et des logiciels peuvent différer de ceux que nous présentons. Ainsi, vérifiez bien votre matériel et la version de vos logiciels avant d'émuler les tutoriels de cette édition spéciale. Vous pouvez installer des versions de logiciels plus récentes ou disponibles dans les dépôts de votre distribution.

Amusez-vous !

Sommaire

Ce mois-ci, nous utiliserons mon capteur de température favori actuellement : le capteur Dallas Semiconductor DS18B20 One Wire. Il ressemble à un transistor « ordinaire », mais c'est un capteur très précis, beaucoup plus que le DHT11 utilisé le mois dernier. Il ne fait pas l'humidité, mais, pour les lectures de température, c'est un composant très bon et peu cher. Toutes les demandes et sorties de données se font sur un seul picot. Il a une plage de fonctionnement de - 55°C à 125°C (-67°F to 257°F) et devrait être capable de fonctionner avec 3 mètres de câble. Il a aussi un mode parasitaire qui permet de dériver l'alimentation de la ligne de données.

La feuille de spécifications peut être trouvée sur : https://datasheets.maxim-integrated.com/en/ds/DS18B20.pdf. Voici à quoi il ressemble :



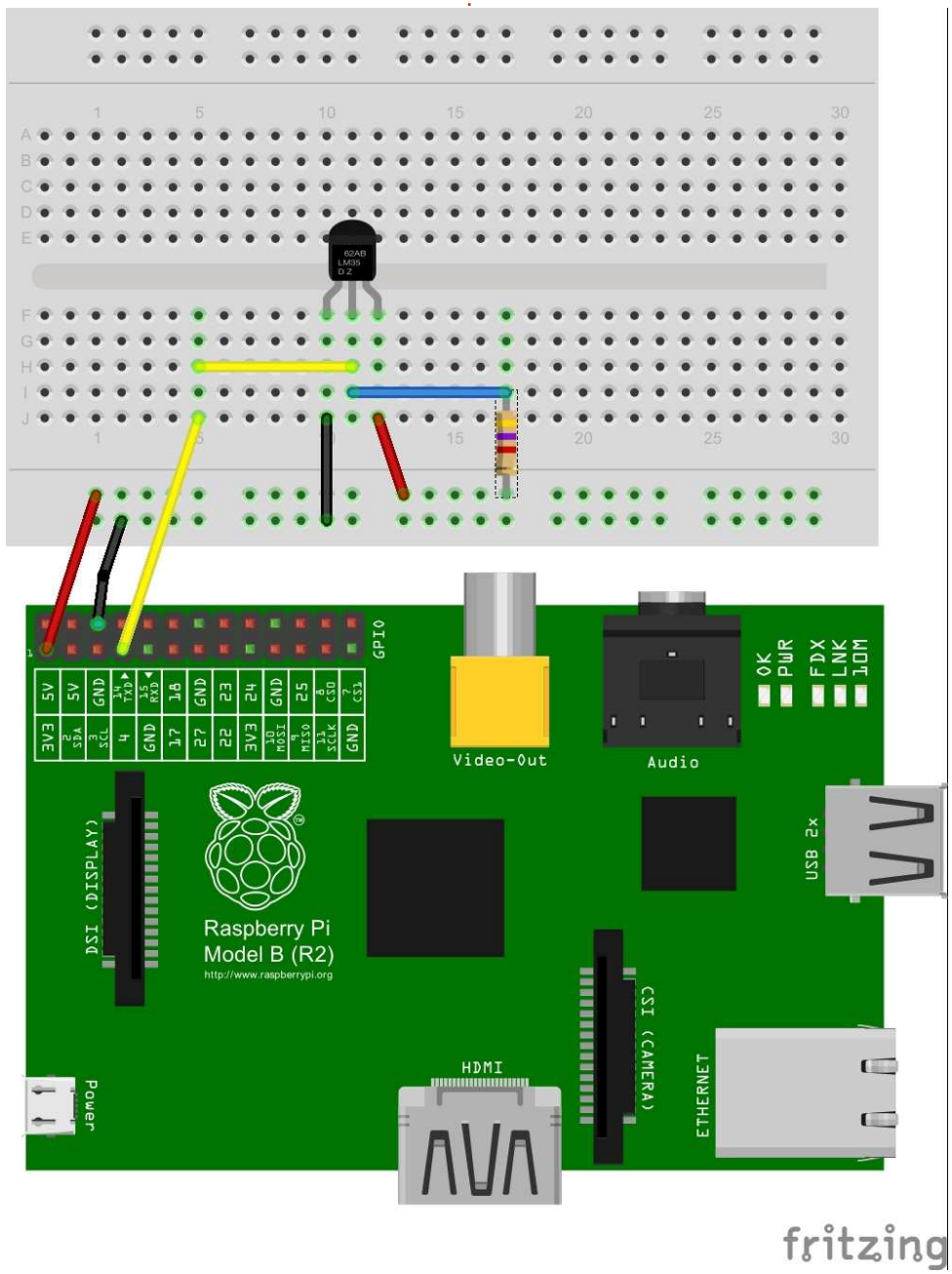
Le câblage d'un seul capteur est facile. Voir le dessin à droite.

Il n'y a que trois connexions au RPi. La masse (picot 1 du capteur) vers le picot 6 du RPi, 3,3 V (picot 3 du capteur) vers le picot 1 du RPi, et les données (picot 2 du capteur) vers le picot 7 du RPi (GPIO 4). Vous avez besoin d'une résistance de 4,7 k entre les picots 2 et 3 du capteur (données et alim +). C'est tout. Si vous souhaitez ajouter d'autres capteurs au projet, il suffit de les connecter masse à masse, alim+ à alim+ et picot 2 à picot 2 du capteur « principal ». Pas besoin de résistances additionnelles pour une longueur de ligne raisonnable. Page suivante, à droite, vous trouvez un exemple d'un projet avec trois capteurs.

Le code

Vous devez dire au système d'exploitation que vous souhaitez utiliser le support de noyau pour les capteurs à un fil. Si vous utilisez Raspbian Jessie, c'est déjà disponible dans raspi-config. Si vous utilisez un autre OS, vous devez alors ajouter la ligne suivante au fichier /boot/config.txt.

```
dtoverlay=w1-gpio
```



Les deux commandes suivantes chargent les pilotes du 1-fil et du thermomètre sur GPIO 4.

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
```

Puis vous devez changer de répertoire (cd) pour le répertoire du composant 1-fil et lister (ls) les appareils pour vérifier que votre thermomètre est chargé correctement.

```
cd /sys/bus/w1/devices/
ls
```

Dans les pilotes des matériels, votre capteur doit être listé comme une série de chiffres et de lettres. Dans mon cas, le composant est enregistré comme 28-000005e2fdc3. Vous devez ensuite accéder au capteur avec la commande cd, en remplaçant mon numéro de série par le vôtre.

```
cd 28-000005e2fdc3
```

Le capteur écrit périodiquement dans le fichier w1_slave ; aussi, nous utilisons simplement la commande cat pour le lire.

```
cat w1_slave
```

Ceci nous amène aux deux lignes de texte suivantes, avec la sortie t= montrant la température en degrés Celsius. Un point décimal devrait être placé avant les trois derniers digits ; par exemple, la température que nous avons reçue se lit 23,125 degrés Celsius.

```
72 01 4b 46 7f ff 0e 10 57 :
crc=57 YES

72 01 4b 46 7f ff 0e 10 57
t=23125
```

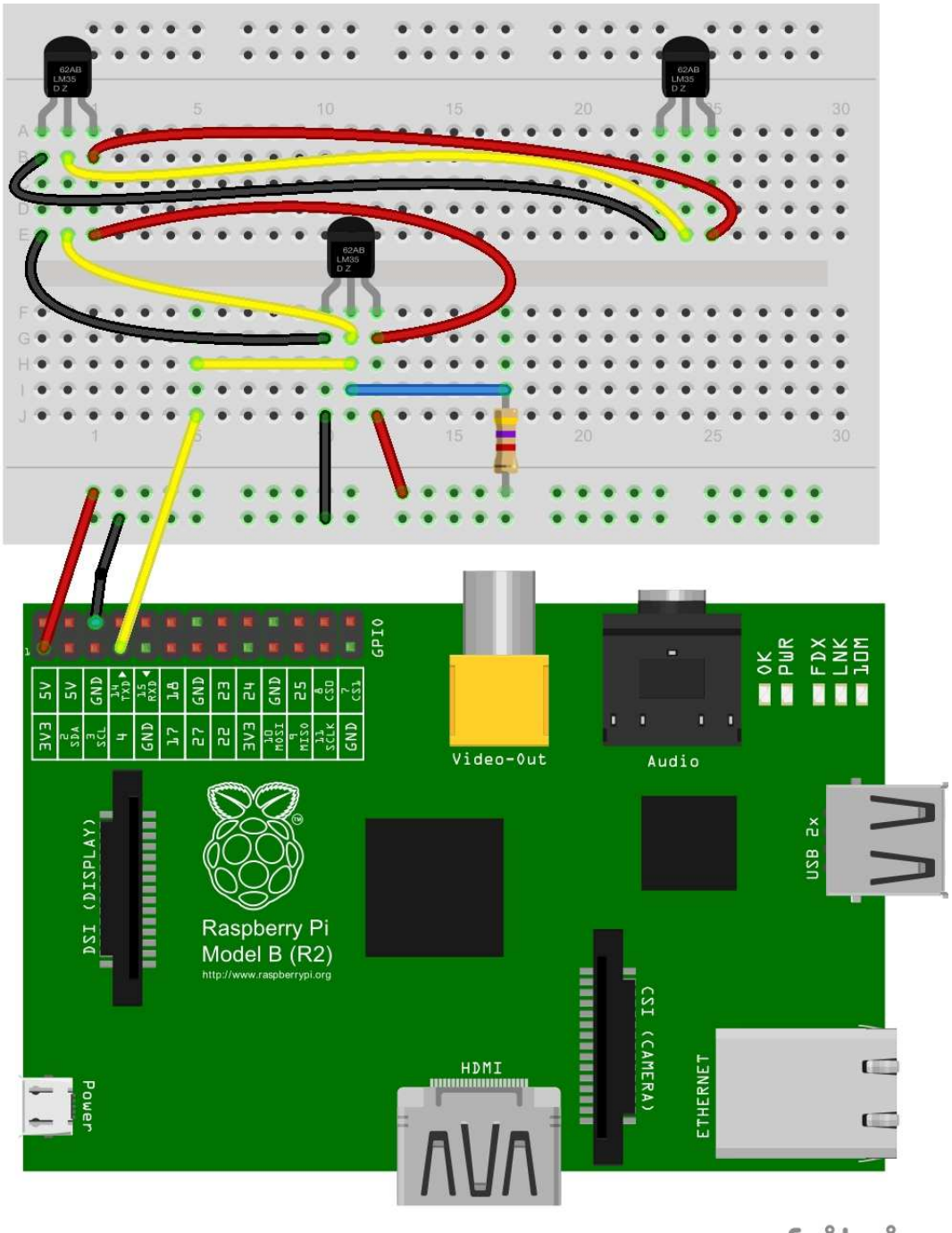
À la page suivante, en haut à droite, vous pouvez voir comment il fallait faire « jadis », comparé à la bibliothèque que nous allons utiliser réellement.

Timo Furrer nous a fourni une bibliothèque fantastique, écrite en pur Python, à utiliser avec le RPi. Vous pouvez la trouver à https://github.com/timofurrer/w1thermsensor. La version actuelle est la 0.3.1 et elle est aussi disponible via pypi.

La beauté de cette bibliothèque vient de ce qu'elle assure presque tout le travail d'échange avec les capteurs et permet de se concentrer uniquement sur le code.

Voyez le code « d'aujourd'hui », page suivante, en bas à droite, utilisant la bibliothèque de Timo…

Seulement 7 lignes de code sont nécessaires ici. Les lignes qui sont commentées vous permettent de voir les



autres façons d'obtenir et d'imprimer les données dans différentes unités de températures (Celsius et Kelvin).

Comme indiqué plus haut, vous pouvez avoir plus d'un capteur sur la même ligne de données. Aussi, voici le code pour un appel unique qui récupère les valeurs de température de tous les capteurs du système.

```
from w1thermsensor import
W1ThermSensor

for sensor in
W1ThermSensor.get_available_s
ensors():

  print("Sensor %s has
temperature %.2f" %
(sensor.id,
sensor.get_temperature()))
```

Bien sûr, vous voudrez faire plus d'un appel de données ; modifiez donc le code ci-dessus de la façon que vous souhaitez.

Si vous voulez faire appel à un capteur en particulier, vous pouvez utiliser ce code comme point de départ.

```
from w1thermsensor import
W1ThermSensor

sensor =

W1ThermSensor(W1ThermSensor.T
HERM_SENSOR_DS18B20, "28-
000007444532")

temperature_in_celsius =
sensor.get_temperature()
```

Ainsi, vous pouvez voir qu'en utilisant la bibliothèque de Timo Furrer vous réduisez votre code de 27 lignes à 3 (pour un seul appel). C'est merveilleux !

Je voulais vous montrer comment utiliser l'afficheur LCD 16×2 avec ceci, mais je pense que je vais laisser la place à d'autres auteurs et nous repousserons cette partie au mois prochain. Ne perdez pas votre matériel de projet, nous l'utiliserons le mois prochain.

En attendant, amusez-vous à vérifier la température de votre bureau/domicile.

```
import os
import glob
import time
os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')
base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'
def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
    return lines
def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f

while True:
        print(read_temp())
        time.sleep(1)
```

```
from w1thermsensor import W1ThermSensor
from time import sleep
sensor = W1ThermSensor()
while 1:
    # temp_in_celsius = sensor.get_temperature()
    temp_in_fahrenheit = sensor.get_temperature(W1ThermSensor.DEGREES_F)
    # temp_in_all_units = sensor.get_temperatures([W1ThermSensor.DEGREES_C, _
          W1ThermSensor.DEGREES_F, W1ThermSensor.KELVIN])
    print temp_in_fahrenheit
    # print temp_in_celsius
    # print temp_in_all_units
    sleep(3)
```

Le mois dernier, nous avons travaillé avec le capteur de température DS18B20. Ce mois-ci, nous commencerons par interfacer un afficheur LCD 16×2 pour montrer nos températures. Ne défaites pas votre montage, mais assurez-vous que vous avez assez de place pour monter votre afficheur sur la plaque d'essai. Vous aurez besoin de 32 trous pour la longueur de la pièce et de 16 trous pour connecter les picots. Vous n'aurez que 3 trous de rab si vous montez l'afficheur en bas des trous verticaux ; aussi, vous aurez besoin de quelques cavaliers pour connecter les trous verticaux du bas à ceux du haut.

Bien sûr, l'afficheur 16×2 a deux lignes de 16 caractères. Le rétro-éclairage existe dans de nombreuses couleurs. J'en ai choisi un bleu. Nous pouvons adresser individuellement chacun des 32 caractères, ou imprimer un peu comme nous le faisons avec le moniteur normal.

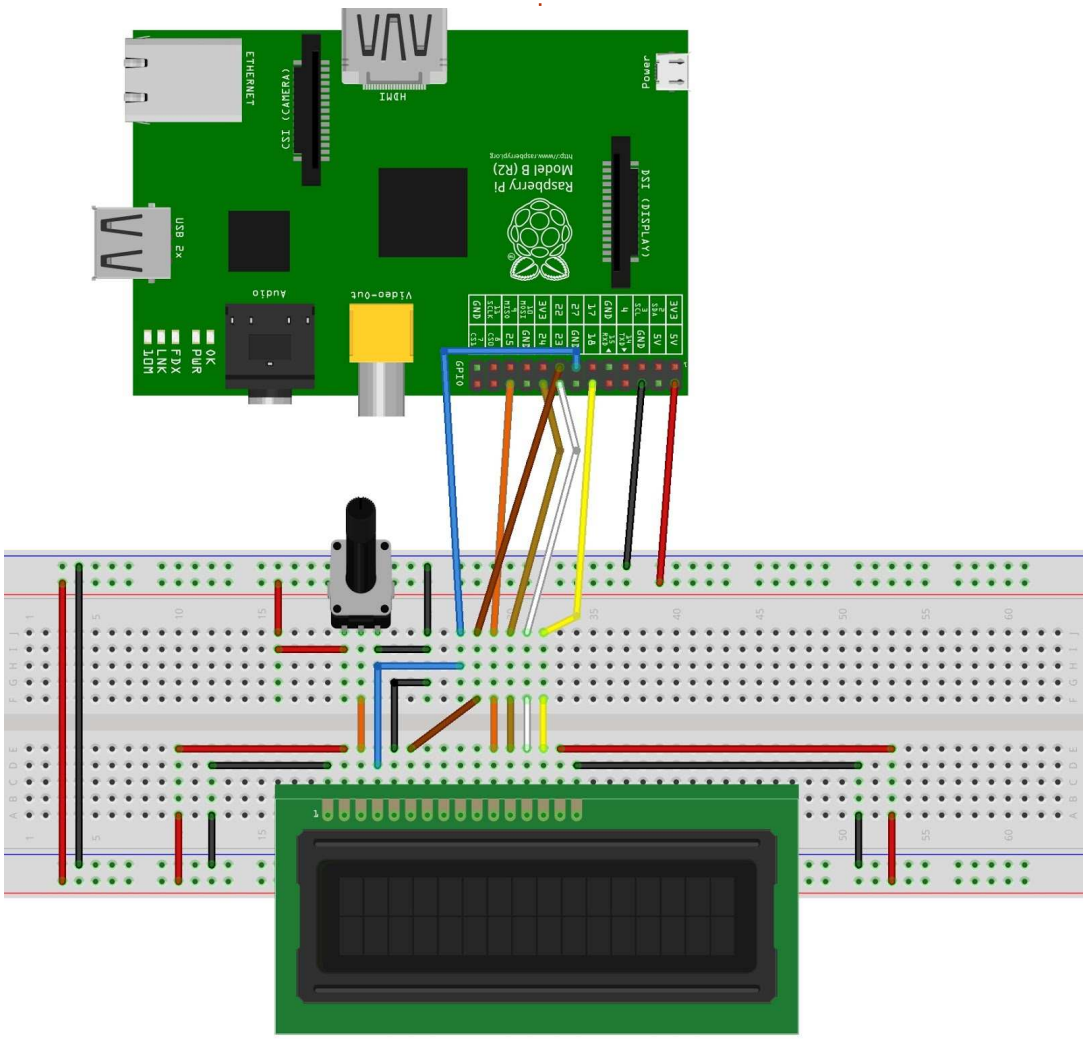
Nous ferons 8 connexions au RPi, en plus des trois que nous utilisions le mois dernier pour le capteur de température. Ce mois-ci, nous aurons besoin des composants supplémentaires suivants :

- un potentiomètre de 10k ;
- un afficheur LCD 16×2 ;
- De nombreux cavaliers pour la plaque d'essai, mâle-mâle et 8 mâle-femelle.

Une fois terminé, le schéma de câblage (et en vrai sur la plaque) ressemblera un peu à un plat de nouilles, mais allez-y lentement, assurez-vous de faire le câblage correctement.

Comme vous pouvez le voir sur le schéma ci-dessus, c'est plutôt horrible ; aussi, je vais vous détailler tout le câblage dans un texte.

D'abord, vous devrez mettre un cavalier entre les deux bus horizontaux, en haut et en bas. Ainsi, vous aurez l'alimentation et la masse sur les deux bus. J'ai choisi de le faire sur la gauche, mais vous pouvez le mettre où vous voulez. Ensuite, câblez le potentiomètre. Un côté (n'importe lequel) doit être à la masse et l'autre à l'alimentation 5 V. Le contact du milieu (le curseur) sera câblé au picot 3 de l'afficheur LCD. Cela contrôle le contraste ; ainsi, vous pouvez régler la luminosité des caractères. Le 5 V, ainsi que la masse, devraient déjà être sur la plaque (cf. le mois dernier).



Sur l'afficheur, connectez le picot 1 à la masse et le picot 2 au bus +5 V. Déjà 3 connexions faites sur les douze. Le picot 6 de l'afficheur est relié au picot 22 du RPi. C'est le picot « Enable » (Activer). Le picot 5 de l'afficheur va à la masse, et le picot 4 au picot 27 du RPi. Nous en sommes à 6 connexions. Nous voici à mi-chemin. Puisque nous devons utiliser le picot 4 pour notre

capteur, nous ne pouvons pas régler le rétro-éclairage.

Maintenant, nous allons travailler en descendant depuis le picot 16. Le picot 16 va à la masse, et le picot 15 au +5 V. Sur le mien, le picot 15 est en fait le voltage du rétro-éclairage. Si vous trouvez l'affichage trop brillant vous pouvez mettre le curseur d'un autre potentiomètre, connecté entre l'alimentation et la masse, pour régler l'intensité de l'affichage.

Maintenant, les lignes de données. Il y a réellement 8 lignes de données, mais, heureusement, nous n'en utiliserons que 4. Les picots 11 à 14 sont D4, D5, D6 et D7. (on compte à partir de 0). Voici le tableau de connexion :

Display Pin	Raspberry Pi Pin
11	25
12	24
13	23
14	18

À présent tout est connecté et nous continuerons avec un échantillon de code pour tester l'afficheur. Mais nous devons obtenir la bibliothèque python Adafruit pour LCD. Dans un terminal, tapez ce qui suit :

```
#!/usr/bin/python
# Example using a character LCD connected to a Raspberry Pi or BeagleBone Black.
import time
import Adafruit_CharLCD as LCD
# Raspberry Pi pin configuration:
lcd_rs      = 27  # Note this might need to be changed to 21 for older revision Pi's.
lcd_en      = 22
lcd_d4      = 25
lcd_d5      = 24
lcd_d6      = 23
lcd_d7      = 18
lcd_backlight = 4
# Define LCD column and row size for 16x2 LCD.
lcd_columns = 16
lcd_rows    = 2
# Alternatively specify a 20x4 LCD.
# lcd_columns = 20
# lcd_rows    = 4
# Initialize the LCD using the pins above.
lcd = LCD.Adafruit_CharLCD(lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6, lcd_d7,
                           lcd_columns, lcd_rows, lcd_backlight)
# Print a two line message
lcd.message('Hello\nworld!')
# Wait 5 seconds
time.sleep(5.0)
# Demo showing the cursor.
lcd.clear()
lcd.show_cursor(True)
lcd.message('Show cursor')
time.sleep(5.0)
# Demo showing the blinking cursor.
lcd.clear()
lcd.blink(True)
lcd.message('Blink cursor')
time.sleep(5.0)
# Stop blinking and showing cursor.
lcd.show_cursor(False)
lcd.blink(False)
# Demo scrolling message right/left.
lcd.clear()
message = 'Scroll'
lcd.message(message)
for i in range(lcd_columns-len(message)):
    time.sleep(0.5)
    lcd.move_right()
for i in range(lcd_columns-len(message)):
    time.sleep(0.5)
    lcd.move_left()
# Demo turning backlight off and on.
lcd.clear()
lcd.message('Flash backlight\nin 5 seconds...')
time.sleep(5.0)
# Turn backlight off.
lcd.set_backlight(0)
time.sleep(2.0)
# Change message.
lcd.clear()
lcd.message('Goodbye!')
# Turn backlight on.
lcd.set_backlight(1)
```

```
git clone
https://github.com/adafruit/A
dafruit_Python_CharLCD

cd Adafruit_Python_CharLCD

sudo python setup.py install

cd examples
```

Maintenant, chargez char_lcd.py dans votre éditeur favori. Ou vous pouvez le copier à la main de la page précédente.

Ignorez les messages de rétro-éclairage ; mais vous devrez voir :

```
Hello World! (Bonjour le
monde)
Show Cursor_ (curseur
visible)
Blink Cursor_ (curseur
clignotant)
Scroll (right and left)
(déplacement, à droite et à
gauche)
Flash backlight in 5 seconds…
(rétro-éclairage flashant
dans 5 secondes)
Goodbye! (Au revoir !)
```

Si tout a bien fonctionné, vous êtes prêt pour un essai. Sinon, revenez en arrière pour contrôler votre câblage.

Voici le programme du mois dernier, modifié, qui inclut des bribes de code venant de cet exemple (en haut à droite) d'Adafruit. (Nouveau code en gras.)

```
from w1thermsensor import W1ThermSensor
from time import sleep
import Adafruit_CharLCD as LCD
# Raspberry Pi pin configuration:
lcd_rs        = 27
lcd_en        = 22
lcd_d4        = 25
lcd_d5        = 24
lcd_d6        = 23
lcd_d7        = 18
lcd_backlight = 4
lcd_columns = 16
lcd_rows    = 2
# Initialize the LCD using the pins above.
lcd = LCD.Adafruit_CharLCD(lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6, lcd_d7,
                           lcd_columns, lcd_rows, lcd_backlight)
sensor = W1ThermSensor()
while 1:
    # temp_in_celsius = sensor.get_temperature()
    temp_in_fahrenheit = sensor.get_temperature(W1ThermSensor.DEGREES_F)
    print temp_in_fahrenheit
    lcd.clear()
    lcd.message(str(temp_in_fahrenheit))
    # print temp_in_celsius
    sleep(3)
```

C'est tout pour cette fois. Le mois prochain, nous examinerons le remplacement de notre afficheur 16×2 classique par un afficheur IC2 16×2 (qui n'utilise que deux lignes pour les données et tout le pilotage et deux lignes pour l'alimentation). Nous présenterons aussi les différentes façons d'utiliser une communication série pour interfacer des afficheurs et autres dispositifs. Jusque-là, amusez-vous bien !

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille.

Ce mois-ci, nous utiliserons un écran LCD 16×2 avec une interface i2c pour faire la même chose que ce que nous avons fait le mois dernier (FCM n° 111) avec le capteur de température Dallas Semiconductor DS18B20. Rappelez-vous, nous avons dû utiliser beaucoup de broches E/S sur le RPi. Cette fois-ci, grâce à l'affichage par i2c, nous aurons seulement besoin de 5 broches (+5v, masse, les données du capteur, SDA (données) et SCL (horloge)) pour faire la même chose.

Avant de commencer notre projet, une présentation de i2c s'impose. J'ai résumé la discussion qui suit d'un merveilleux tutoriel écrit par ***Sfuptownmaker*** sur Sparkfun.com accessible à https://learn.sparkfun.com/tutorials/i2c?_ga=1.242063243.863781319.1463423290

I2C

Le Protocole Circuits Inter-Intégrés (i2c - Inter-Integrated Circuits) est destiné à permettre à plusieurs CI numériques « esclaves » de communiquer avec une ou plusieurs puces maîtres. Tout comme SPi (Serial Peripherical Interface - Périphérique à Interface Série), il est prévu pour des communications à courte distance sur un seul appareil. Comme l'Interface Série Asynchrone (RS-232 ou UART), il ne nécessite que deux câbles de signaux pour échanger des informations.

RS232 (Communication Asynchrone)

Aucune donnée d'horloge n'est nécessaire sur les lignes ; cependant, les deux côtés doivent s'accorder sur le débit de données de communication. Il nécessite un surcoût matériel (UART) à chaque extrémité.

Outre les données sur 8 bits, au moins un bit de début et un bit d'arrêt sont nécessaires pour chaque trame de données. Bien qu'il soit possible de connecter plusieurs périphériques sur un seul port série, si plusieurs périphériques tentent d'utiliser les deux lignes simultanément, des problèmes s'ensuivent.

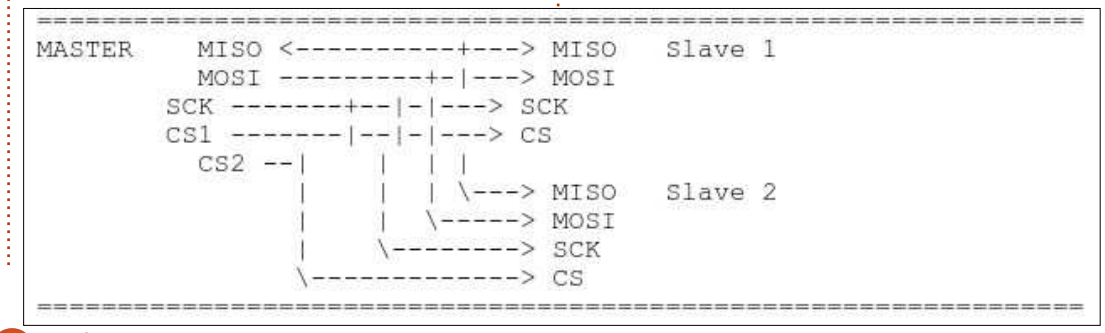
La plupart des dispositifs UART ne peuvent supporter que certaines vitesses de transmission pré-réglées, le maximum habituel est de 230 400 bits par seconde.

```
RX <--------- TX
TX ---------> RX
```

SPI

Le plus gros inconvénient de SPI est le nombre de picots nécessaires. La connexion d'un seul esclave à un unique maître requiert quatre lignes, chaque esclave supplémentaire nécessite l'addition d'un picot « chip select » (CS - sélection de circuit intégré) d'entrée/sortie sur le maître. Si vous voulez utiliser plusieurs capteurs/dispositifs connectés au maître, le nombre nécessaire de picots peut rapidement dépasser le nombre des entrées/sorties disponibles.

SPI (voir diagramme ci-dessous) est bon pour un taux de transfert élevé en full-duplex (envoi/réception simultanés de données). Le taux de transfert de données peut monter jusqu'à 10 Mhz.



I2C

Comme un port série asynchrone, i2c ne nécessite que deux lignes, mais ces deux lignes peuvent supporter jusqu'à 1 008 périphériques esclaves. Contrairement à SPI, i2c peut prendre en charge un système multi-maître, permettant à de multiples périphériques esclaves de communiquer avec plusieurs périphériques maîtres. Les maîtres ne peuvent pas communiquer entre eux sur le bus i2c et doivent utiliser les lignes de bus à tour de rôle ; il y a donc des limitations. i2c a un « coût » semblable à celui d'Async en ce sens que, pour chaque donnée de 8 bits, un bit supplémentaire est nécessaire comme un bit "Ack/Nack". Les exigences matérielles sont plus complexes que le SPI, mais inférieure à Async.

Les débits de données se situent entre Async et SPI. La plupart des

```
=============================================================================
Master 1     SDA <------+-----------------------> SDA    Slave 1
             SCL -------|-------------+---------> SCL
                        |             |
Master 2     SDA <------+-------------|---------> SDA    Slave 2
             SCL ---------------------+---------> SCL
=============================================================================
```

périphériques i2c peuvent communiquer entre 100 KHz à 400 KHz.

Dans le schéma ci-dessus, SDA est la ligne de données et SCL est la ligne d'horloge.

Espérons que je ne vous aie pas totalement embrouillé et que vous êtes prêt à continuer avec notre projet.

Une très bonne ressource pour ce que nous sommes sur le point de le faire est sur http://www.circuitbasics.com/raspberry-pi-i2c-lcd-set-up-and-programming/

Assurez-vous que i2c est activé sur le RPi. On règle cela dans raspi-config.

Maintenant, dans un terminal, utilisez apt-get pour installer deux bibliothèques de support. (Je n'ai pas réussi à le lancer sur une seule ligne pour une raison inconnue.) :

```
sudo apt-get install i2c-tools

sudo apt-get install python-smbus
```

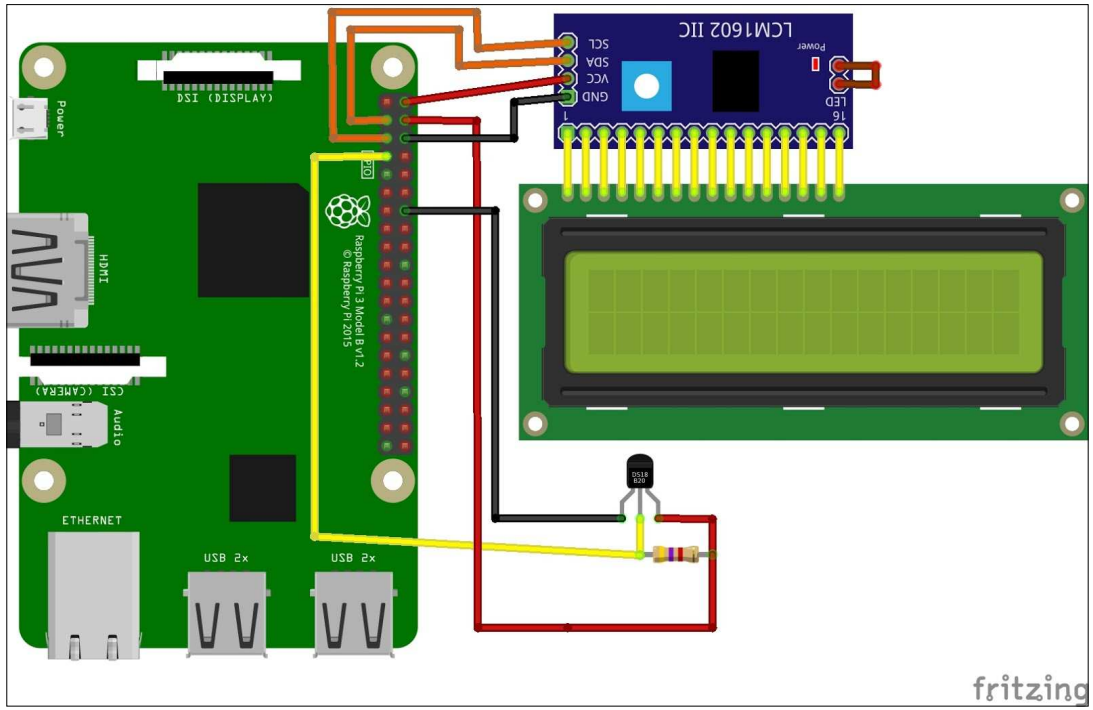
J'ai pu obtenir le schéma Fritzing équipé avec l'adaptateur i2c (ci-contre).

Branchez la broche SDA de l'adaptateur i2c à la broche PHYSIQUE n° 3 sur le RPI (qui est GPIO2) et le SCL de l'adaptateur à la broche PHYSIQUE n° 5 (GPIO3). Choisissez une broche de 5V libre sur le RPi (broche 2 ou 4) et une masse libre (broche 6 ou 9) et connectez-les aux VCC (+5V) et Gnd (masse) de l'adaptateur. N'oubliez pas que nous devons connecter le capteur de température à GPIO4, comme le mois dernier (avec la résistance à +5VDC).

Maintenant, redémarrez et, une fois que le RPi est prêt, tapez dans le terminal :

```
i2cdetect -y 1
```

Ceci permettra de vérifier que l'interface i2c fonctionne sur votre Pi, et vous dira aussi quelle est l'adresse utilisée par l'afficheur LCD. Regardez la capture d'écran montrée à droite pour voir ce à quoi il devrait ressembler.




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

Comme vous pouvez le voir, mon appareil est à 3f, mais le vôtre pourrait être à une adresse différente. Lorsque vous créez le pilote ci-dessus (en tapant, soit directement à partir de l'article, soit de la page de pastebin), vous devrez entrer l'adresse de votre appareil à la ligne 22.

La première série de code est une bibliothèque qui fonctionnera comme un pilote pour le LCD i2c. Cela devrait

être enregistré sous i2c_lcd_driver.py. Le code est disponible sur http://pastebin.com/ueu18fNL pour vous épargner la saisie.

Maintenant, nous allons faire un petit test pour nous assurer que tout fonctionne. Tapez le code suivant et enregistrez-le sous i2c_test1.py dans le même dossier que le pilote que nous venons d'écrire :

```
import i2c_lcd_driver from
time import
mylcd = i2c_lcd_driver.lcd()
mylcd.lcd_display_string("Ceci
est un test",1)
```

Si tout est correct, vous devrez voir « Ceci est un test » à la position du premier caractère de la première ligne de l'écran LCD. Si c'est bon, nous pouvons continuer. Le code suivant est pratiquement le même que le mois dernier, modifié pour utiliser l'affichage par i2c au lieu de la version parallèle.

Cela termine notre présentation des écrans LCD et i2c. Nous utiliserons les écrans LCD i2c et d'autres dispositifs i2c à l'avenir ; vous devrez donc les garder en sécurité pour plus tard. Le mois prochain, nous allons commencer à travailler avec des moteurs, des servos et des moteurs pas-à-pas. Alors, dépêchez-vous de vous procurer un « hobby motor » (un moteur miniature

```
import i2c_lcd_driver
from w1thermsensor import W1ThermSensor
from time import *

mylcd = i2c_lcd_driver.lcd()

#mylcd.lcd_display_string("This is a test",1)

sensor = W1ThermSensor()
#setup_lcd()
while 1:
      # This is basically the same code as last month, so use
      # whichever temp type you want.
    temp_in_fahrenheit = sensor.get_temperature(W1ThermSensor.DEGREES_F)
    # Print the temp to the terminal...
    print temp_in_fahrenheit
    # Now print it to the i2c LCD module...
    mylcd.lcd_clear()
    mylcd.lcd_display_string(str(temp_in_fahrenheit),1)
    sleep(3)
```

de loisirs) pour être prêt. Dans quelques mois, nous allons commencer à travailler avec le microcontrôleur Arduino, puis apprendre à interfacer le RPi pour contrôler l'Arduino.

Jusque-là, amusez vous bien.

Greg Walters est propriétaire de Rainy-Day Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille.

Ce mois-ci, nous allons utiliser le RPi pour contrôler un simple moteur de loisirs à courant continu (DC). On peut se procurer celui-ci dans la plupart des boutiques de loisirs, des fournisseurs d'électronique et même dans certaines grandes chaînes de quincaillerie. Voici une « liste de courses » énumérant ce dont nous aurons besoin :

- Moteur de loisirs DC.
- Puce de contrôle moteur à double pont en H L293D.
- 4 piles AA (ou AAA) et un support pour piles.
- Planche à essai.
- Des cavaliers mâle-mâle.
- Le RPi (bien entendu).

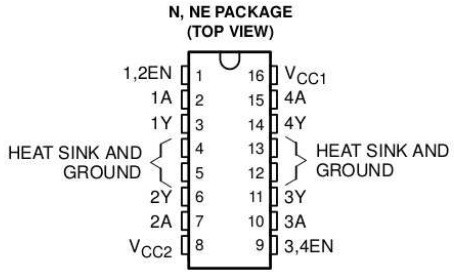
Avant de commencer le câblage et le codage, nous devons parler de deux ou trois choses.

Tout d'abord, ne JAMAIS, JAMAIS, connecter un moteur de n'importe quel type directement au RPi. C'est une catastrophe assurée. Les besoins en courant peuvent carrément faire « fondre » le RPi. La puce de contrôle ne coûte même pas 5 $ US, ce qui est beaucoup moins cher qu'un RPi à 39 $.

Ensuite, nous discuterons du pilote de moteur L293D à pont en H pendant quelques instants pour que vous puissiez comprendre le fonctionnement de ce dispositif.

Selon Wikipedia, *« Un pont en H est un circuit électronique qui permet l'application d'une tension sur une charge dans les deux sens. Ces circuits sont souvent utilisés dans la robotique et d'autres applications pour permettre aux moteurs DC de tourner en avant et en arrière. »*

Voici le brochage de la puce de pilotage (« emprunté » auprès de hardwarefun.com)…



Les broches 1 et 9 sont des broches enable (activation). Considérez-les comme des interrupteurs On/Off. Un état bas de la broche enable signifie que le moteur est éteint. Un état haut signifie que le moteur PEUT ÊTRE allumé. Regardons cela comme un tableau logique ou une table de vérité. Les broches 1A et 2A sont sur un côté de la puce et sont des lignes de contrôle comme les broches enable. La même logique s'applique aussi à 3A et 4A (l'autre moitié de la puce). Les broches 1Y et 2Y sont les sorties vers le moteur.

Enable	1A	2A	Result
LOW	HIGH	LOW	Not spinning - Enable is “off”
LOW	LOW	HIGH	Not spinning - Enable is “off”
HIGH	LOW	LOW	Not spinning - Both control inputs are “Off”
HIGH	HIGH	LOW	Turning Clockwise*
HIGH	LOW	HIGH	Turning Anti-Clockwise*
HIGH	HIGH	HIGH	Not Spinning - Both control inputs are “On”
			* Direction is based on motor wiring.

Le résultat du tableau dément ci-dessus est celui-ci : si vous voulez que le moteur s'allume :

- l'état de la broche enable DOIT être HAUT (la broche 1 et/ou la broche 9),
- ET L'ÉTAT DE, SOIT 1A, SOIT 2A, MAIS PAS LES DEUX, DOIT être HAUT (la broche 2 et la broche 7 respectivement).

Ayant décodé la logique de la puce magique, nous pouvons commencer le câblage de la plaque d'essai et du RPi.

Câblage

Le dessin Fritzing (en haut à droite de la page suivante) montre le diagramme de câblage pour ce mois-ci. Remarquez que nous n'utilisons qu'une moitié de la puce, ce qui veut dire que nous pourrions, en fait, contrôler deux petits moteurs DC et pas seulement un. C'est à vous d'expérimenter cela !

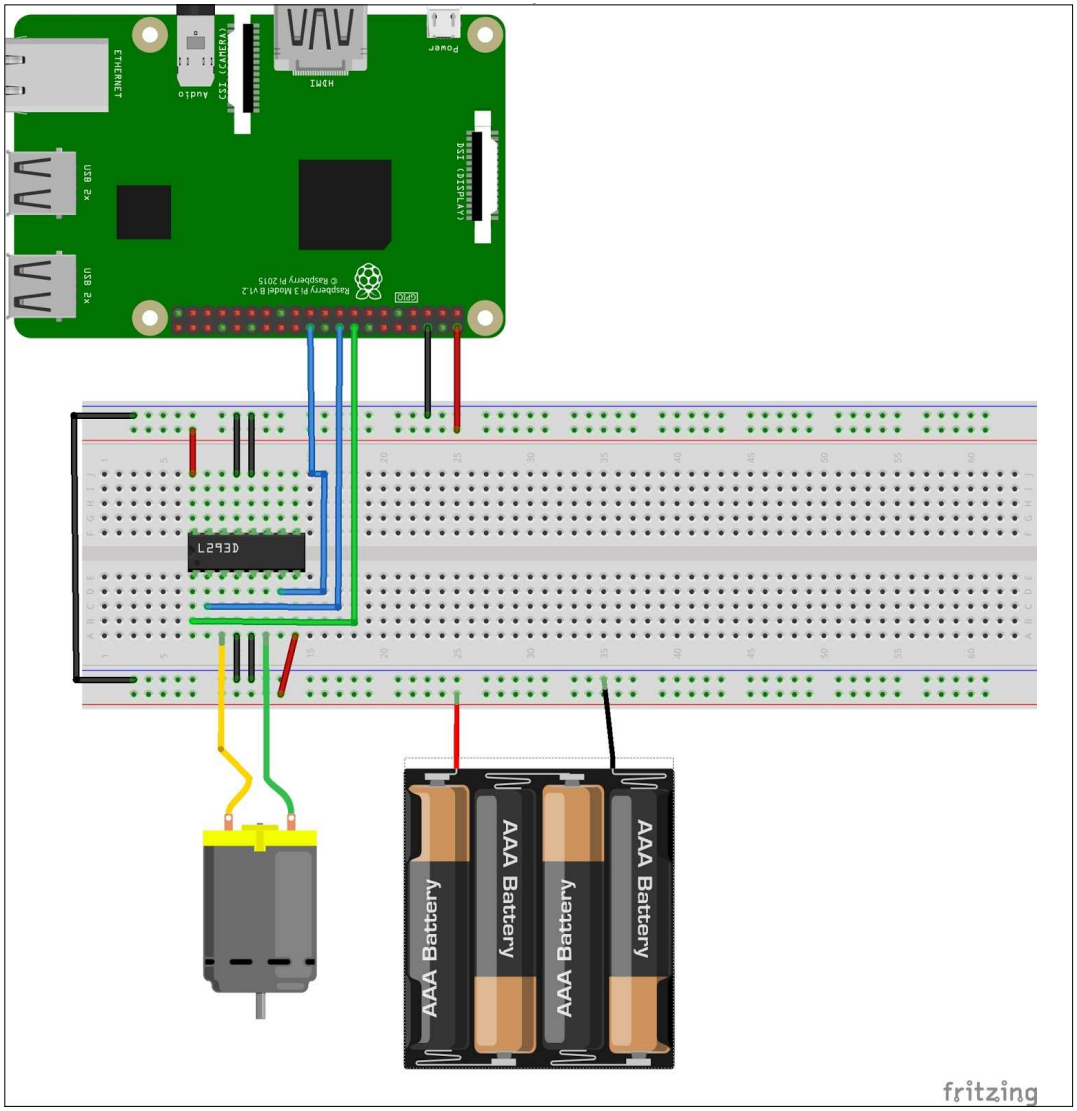
Comme toujours, connectez les câbles au RPi AVANT de l'allumer. En outre, il faut vérifier et revérifier le câblage, surtout à cause de l'alimentation externe. Vous pourriez le regretter vivement si quelque chose était sur la mauvaise broche.

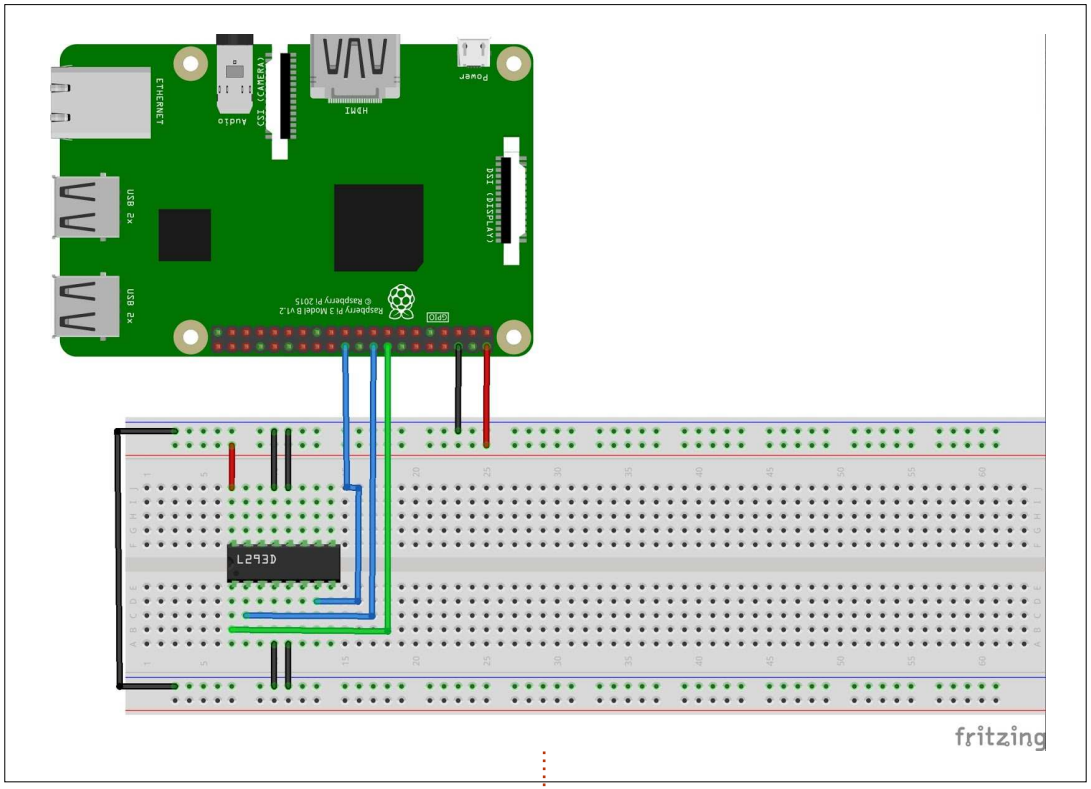
Cette première image Fritzing montre les connexions au RPi et à la plaque d'essai/la puce. C'est distribué

comme le montre le tableau en bas à droite.

Le diagramme Fritzing suivant (ci-dessous) montre les connexions entre la batterie et le moteur.

Nous utilisons l'alimentation +5 VDC du RPi pour faire tourner la puce de contrôle du moteur (RPi broche 2 vers L293D broche 16). Le diagramme ci-dessus montre des piles AAA, mais vous pouvez aussi vous servir d'un bloc piles qui utilise des piles AA. Nous fournissons la masse du RPi (broche 6) vers la puce (broches 4,5,12,13). Ce





FROM		TO	
RPi	Pin 2 (+5VDC)	Breadboard	VCC Rail
RPi	Pin 6 (Gnd)	Breadboard	Gnd Rail
RPi	Pin 16 (GPIO 23)	Chip	Pin 1 (Enable)
RPi	Pin 18 (GPIO 24)	Chip	Pin 2 (1A)
RPi	Pin 22 (GPIO 25)	Chip	Pin 7 (2A)
Chip	Pins 4,5,12,13	Breadboard	Gnd Rail
Chip	Pin 16	Breadboard	VCC Rail
Chip	Pin 8	Breadboard	BOTTOM VCC Rail
Chip	Pins 12,13	Breadboard	BOTTOM Gnd Rail
Breadboard	Gnd Rail Top	Breadboard	Gnd Rail Bottom

sont les broches 3 (1A) et 5 (2A) de la puce qui font tourner le moteur. La pile se connecte à la broche 8 de la puce afin de fournir la tension au moteur.

Code

Nous allons traiter le code dans deux programmes. Le premier allume le moteur, le laisse tourner pendant quelques secondes, puis l'arrête. Le deuxième est une version modifiée du premier qui démontre comment inverser le sens du moteur.

Dcmotor1.py

Ce programme (ci-dessous) allumera tout simplement le moteur en marche avant (dans le sens des aiguilles d'une montre), le laissera tourner, puis l'arrêtera. Essentiellement, il démontrera que tout fonctionne comme il faut.

```
import RPi.GPIO as GPIO
```

from time import sleep

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(23,GPIO.OUT)   # 1A
GPIO.setup(24,GPIO.OUT)   # 2A
GPIO.setup(25,GPIO.OUT)   # Enable
GPIO.output(24,GPIO.LOW)
```

Set everything up and set 2A to low.

```
print "Starting motor"
GPIO.output(23,GPIO.HIGH)
GPIO.output(25,GPIO.HIGH)

sleep(5)
```

Set 1A to HIGH and Enable to HIGH to start the motor and let it run for 5 seconds.

```
print "Stopping motor"
GPIO.output(25,GPIO.LOW)
sleep(2)
GPIO.cleanup()
```

Stop the motor by setting the Enable to LOW, sleep for 2 seconds, then run GPIO.cleanup().

The first part of the program will be used in the next one.

Dcmotor2.py

Dans ce programme (page suivante), nous réglons les broches GPIO comme nous l'avons fait auparavant, mais maintenant, nous utilisons PWM (modulation de largeur d'impulsion) pour moduler la vitesse du moteur. Si vous ne vous souvenez pas de PWM, veuillez revoir la partie 64, dans le FCM n° 107.

En marche avant, plus le rapport cyclique est long (plus près de 100), plus le moteur tournera vite.

En marche arrière, plus le rapport cyclique est COURT (plus près de 0), plus vite tournera le moteur.

Nous accélérons le moteur en réglant le rapport cyclique sur un FAIBLE pourcentage, nous le laissons tourner pendant 5 secondes, puis nous l'arrêtons, faisons un GPIO.cleanup(), et terminons le programme.

C'est tout pour ce mois-ci. Le mois prochain, nous allons travailler avec des servos. Tout ce dont vous aurez besoin est un petit servo peu cher avec trois fils. Nous n'utiliserons pas de pièces du projet actuel, mais gardez-les pour des projets futurs.

Jusque-là, amusez-vous bien.

Greg Walters est propriétaire de RainyDay Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille.

```
import RPi.GPIO as GPIO
```

from time import sleep

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(23,GPIO.OUT)  # 1A
GPIO.setup(24,GPIO.OUT)  # 2A
GPIO.setup(25,GPIO.OUT)  # Enable
GPIO.output(24,GPIO.LOW)
```

As I stated earlier, the above code is pretty much the same thing as we started with in dcmotor1.py.

```
fwd = GPIO.PWM(23,40)
```

We are setting pin 23 to be a PWM Output line with 40% duty cycle (on 40% of the time and off 60% of the time.

```
print "Starting motor"
GPIO.output(25,GPIO.HIGH)
fwd.start(70)
sleep(5)
```

We start the motor by setting the enable to High and setting the Duty Cycle to 70. The motor will run for 5 seconds.

```
print "Stopping motor"
GPIO.output(25,GPIO.LOW)
sleep(2)
```

Now, we stop the motor by setting enable to low.

```
print "Starting motor in reverse"
rev = GPIO.PWM(24,50)
GPIO.output(23,GPIO.LOW)
GPIO.output(25,GPIO.HIGH)
rev.start(50)
sleep(5)
```

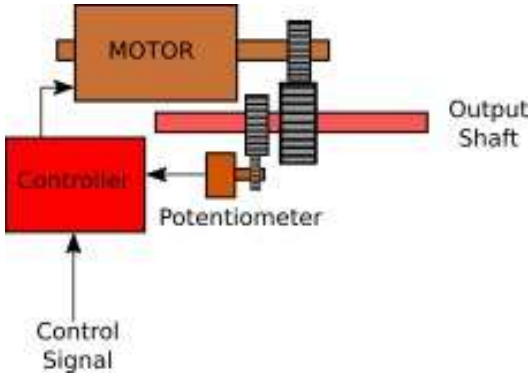
We now set the motor to reverse (pin 23 to low and starting the PWM duty cycle to 50% and run for 5 seconds…

```
print "Speeding up the motor..."
rev.ChangeDutyCycle(10) # When reversing the motor, a smaller duty
                        # Cycle means faster.
sleep(5)
print "Stopping motor"
GPIO.output(25,GPIO.LOW)
GPIO.cleanup()
```

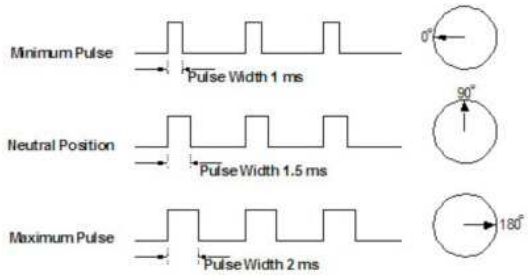
Python dans le monde réel - P. 71

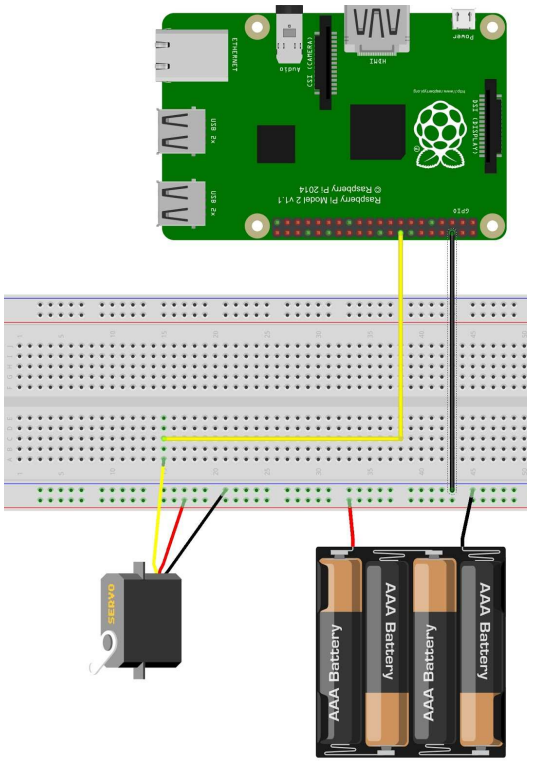
Ce mois-ci, nous allons interfacer un servo-moteur à notre RPi. Il ne faut qu'un servo-moteur, le Raspberry Pi, la plaque d'essai, quelques cavaliers et le bloc de piles utilisées le mois dernier.

Un servo-moteur est simplement un moteur qui a un circuit de commande et un potentiomètre pour donner au circuit de commande la position de l'axe de sortie. La PLUPART des servos feront tourner leur axe entre 0 ° et 180 °. Quelques-uns vont jusqu'à 360 °, voire plus, mais ils coûtent cher. Des engrenages font toutes les liaisons entre le moteur, le potentiomètre et l'axe de sortie. Nous fournissons l'alimentation par des batteries ou une autre source extérieure d'alimentation et le RPi enverra les signaux de commande. La plupart des servos ont seulement 3 fils, tension positive, tension négative (masse) et signal de commande. La couleur des fils varie d'un fabricant à l'autre, mais les fils d'alimentation devraient être de couleurs proches du rouge et du noir et le fil de commande est celui qui reste. En cas de doute, vérifiez la feuille de données du fabricant.



Les signaux de commande sont attendus sous une « forme » très spécifique et nous utiliserons PWM (Pulse Width Modulation - Modulation de largeur d'impulsion) pour cela. D'abord, les impulsions doivent arriver toutes les 20 millisecondes. La largeur de l'impulsion détermine dans quel sens tourne l'axe de sortie. Si l'impulsion est large de 1 ms, le moteur tournera jusqu'à 0 °. Si l'impulsion est de 1,5 ms, alors l'axe tourne jusqu'à 90 °. Si l'impulsion est de 2 ms, l'axe tourne jusqu'à 180 °.





Le câblage

Les connexions sont très simples ce mois-ci. Le bloc de piles alimente le moteur, de sorte que la tension + sur le servo aille sur la ligne + et que le fil négatif du servo aille sur la ligne -. Nous connectons la tension négative du bloc de piles (ligne négative) sur le picot 6 du RPi. Le picot 23 de GPIO (picot 16) est relié au fil de commande du servo.

Maintenant, un peu de maths. Comme nous avons dit précédemment, le servo attend un signal toutes les 20 ms pour fonctionner, et nous devons envoyer en permanence ces impulsions pour garder l'axe de sortie dans la position que nous voulons. La commande GPIO pour paramétrer la modulation de largeur d'impulsion (PWM) est :

```
Pwm = GPIO.pwm({RPi
Pin},{Frequency})
```

Nous connaissons le numéro du picot (23), mais nous devons convertir les 20 ms en Hertz pour paramétrer la commande de réglage de pwm. Comment le faire ? C'est simple.

```
Fréquence = 1/temps
Fréquence = 1/0,02 (20 ms)
Fréquence = 50 Hertz
```

Ainsi, maintenant, quand nous préparons notre code, nous pouvons régler la commande GPIO.pwm pour le picot de commande et utiliser 50 pour notre fréquence.

Notre premier programme commencera au voisinage de 0 ° et se déplacera à proximité de 90 ° puis ira jus-

qu'à 180 ° environ. Je dis seulement proche, parce que chaque servo est un peu différent. Nous ne voulons pas régler le rapport cyclique à 0 et l'entendre claquer en butée avec une détérioration éventuelle du servo ; aussi, nous commencerons avec un petit nombre, proche de 0, et terminerons avec un nombre proche de 12 de façon à commencer à « composer » un ensemble de valeurs qui fonctionnent. Pour mon servo, les nombres 3 pour 0 ° et 12 pour 180 ° marchent bien.

Servo1.py

```
import RPi.GPIO as GPIO from
time import sleep

GPIO.setmode(GPIO.BCM)
GPIO.setup(23,GPIO.OUT)

pwm = GPIO.PWM(23,50)
pwm.start(2)
pwm.ChangeDutyCycle(3)
sleep(5)
```

3 vous donnera le premier angle, mais vous pourriez avoir à essayer avec 2, 1, 0 ou 4 pour y parvenir. Notez bien ce numéro.

```
pwm.ChangeDutyCycle(6)

sleep(5)
```

Ceci devrait placer le rotor en position centrale (90 °). Si vous avez modifié le premier chiffre ou le suivant, celui-ci devra aussi être changé.

```
pwm.ChangeDutyCycle(12)

sleep(5)
```

Le dernier chiffre devrait vous amener à la position 180 °. Là encore, essayez quelques chiffres d'un côté ou de l'autre pour l'ajuster, et une fois fait, notez la valeur.

```
GPIO.cleanup()
```

Enfin, nous appelons GPIO.cleanup() pour un retour à la normale.

Maintenant arrive le gros morceau de maths. Nous utiliserons les valeurs 3 et 12 pour y1 et y2 respectivement dans la formule. Remplacez-les par vos propres numéros.

```
Offset = (y2-y1)/(x2-x1)
Offset = (12-3)/(180-0)
Offset = 9/180
Offset = .05
```

Maintenant, si nous réglons le rapport cyclique pour un angle entre 0 ° et 180 °, nous utilisons la formule suivante :

```
DutyCycle = (Offset * angle)
+ 2.0

DutyCycle = (.05 * angle) +
2.0
```

Quand je l'ai fait, ça fonctionnait, mais j'ai trouvé que la valeur de 0,061 marchait un petit peu mieux.

Servo2.py

```
import RPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)
GPIO.setup(23,GPIO.OUT)
pwm = GPIO.PWM(23,50)
pwm.start(2)
def SetAngle(angle):
        DutyCycle=(.061 * angle) + 2.0
        print(DutyCycle, angle)
        pwm.ChangeDutyCycle(DutyCycle)

try:
        while True:
                for a in range(0,180):
                        SetAngle(a)
                        sleep(.05)

                for a in range(180,0,-1):
                        SetAngle(a)
                        sleep(.05)

except KeyboardInterrupt:
        GPIO.cleanup()
```

Voilà, c'est tout pour ce mois-ci. La prochaine fois, nous travaillerons avec un moteur pas-à-pas, une sorte de croisement entre un servo et un moteur ordinaire.

Jusque-là, amusez-vous bien !

Greg Walters est propriétaire de RainyDay Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille.

Bienvenue de nouveau dans ce que j'appelle affectueusement « La folie de Greg pour Python ». Comme promis, nous allons travailler à la connexion d'un moteur pas-à-pas à un Raspberry Pi. Vous aurez besoin de votre Raspberry Pi, un moteur pas-à-pas de loisirs, 4 piles AA et le bloc-piles, la puce de commande L293D que nous avons utilisée précédemment, une plaque d'essais, et quelques cavaliers.

Alors que je faisais des recherches pour ce projet-ci, je suis tombé sur un guide sur tutorials-raspberrypi.de. J'ai été tellement impressionné par les informations de ce site web que j'utilise une majorité de leurs renseignements et de leur code dans cet article. Le site Web : http://tutorials-raspberrypi.com/how-to-control-a-stepper-motor-with-raspberry-pi-and-l293d-uln2003a/. Si vous êtes perdu dans mes explications, vous pouvez toujours y aller et obtenir quelques éclaircissements.

Le moteur choisi est un mini-moteur pas-à-pas Radio Shack. En fait, c'est un moteur basse tension 28BJY-48. Avant d'essayer de connecter un moteur pas-à-pas, merci de vous procurer sa documentation et autant d'informations que possible. Pour mon cas, la documentation est ici : http://www.tutorials-raspberrypi.de/wp-content/uploads/2014/08/Stepper-Motor-28BJY-48-Datasheet.pdf

Bon, regardons d'abord les moteurs pas-à-pas en général et, ensuite, nous préciserons cette information pour le 28BJY et nous le connecterons au Pi via notre puce de commande L293D.

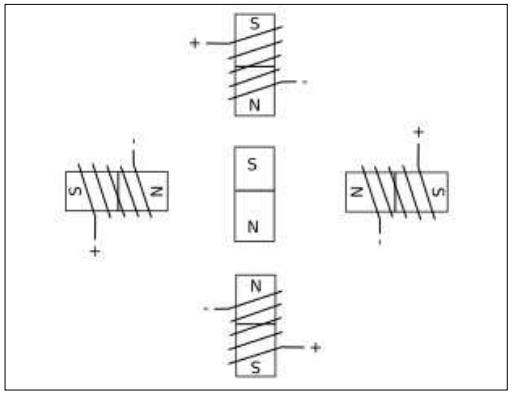
Les moteurs pas-à-pas

Les moteurs pas-à-pas sont utilisés en robotique et dans les machines à commande numérique lorsque vous voulez pouvoir facilement bouger un objet vers un endroit précis. Il y a deux types de base de moteurs pas-à-pas, l'un appelé unipolaire et l'autre bipolaire. Leur différence apparaîtra plus loin dans ce tutoriel. Le 28BJY est bipolaire et a aussi une boîte de vitesses.

Dans les deux cas, il y a un certain nombre de bobines électromagnétiques qui sont allumées et coupées successivement pour faire tourner le moteur. Chaque fois que nous alimentons l'une des bobines, le moteur tourne un petit peu (si alimenté dans la séquence correcte pour le moteur). On appelle cela un pas, d'où le nom de moteur pas-à-pas.

Moteurs unipolaires

Les moteurs unipolaires possèdent des bobines alimentées dans une seule direction : le UNI de unipolaire. Le rotor du moteur est contrôlé en alimentant, puis coupant, les différentes bobines dans une séquence spécifique pendant un certain temps. Voici un diagramme simplifié de ce modèle :

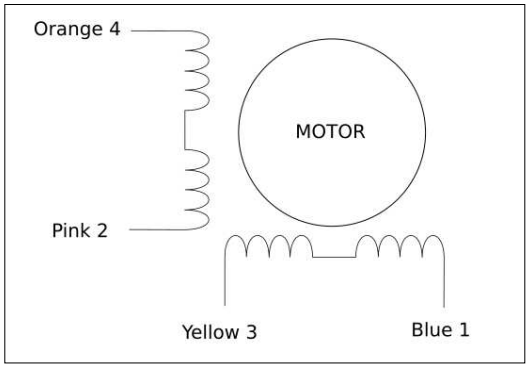


Alimenter chaque bobine, une à la fois, va faire que l'aimant du rotor va se tourner vers la bonne bobine. Si on se guide sur un cadran d'horloge, alimenter les bobines successivement à 12 h, 3 h, 6 h, 9 h, puis de nouveau 12 h va faire tourner le rotor d'un tour complet dans le sens des aiguilles d'une montre. On a donc besoin de quatre « pas » pour faire un tour. On appelle cela l'onde unipolaire. En allant plus loin, nous pourrions avoir un mouvement plus granulaire en activant alternativement les bobines, mais en activant aussi la bobine d'après, ce qui fait faire un huitième de tour au rotor quand les deux bobines sont alimentées. La suite serait alors : 12, 12 et 3, 3, 3 et 6, 6, 6 et 9, 9, 9 et 12, et finalement 12 de nouveau seule. On a ainsi 8 pas par tour, ce que l'on appelle le fonctionnement en demi-pas. Pour faire aller le moteur en marche arrière (sens inverse des aiguilles), nous inversons simplement la séquence. Ceci est une représentation TRÈS simple, et beaucoup de moteurs pas-à-pas ont une résolution qui peut aller jusqu'à 200 pas par tour.

Moteurs bipolaires

Le 28BJY, comme dit précédemment, est un moteur bipolaire. Dans ce cas, les bobines peuvent voir leur courant inversé et deux bobines sont alimentées en même temps. Cela crée une situation où la commutation est

plus complexe, mais la quantité de force de rotation (puissance) du rotor est augmentée. Voir ci-dessous un diagramme bloc simple du 28BJY.

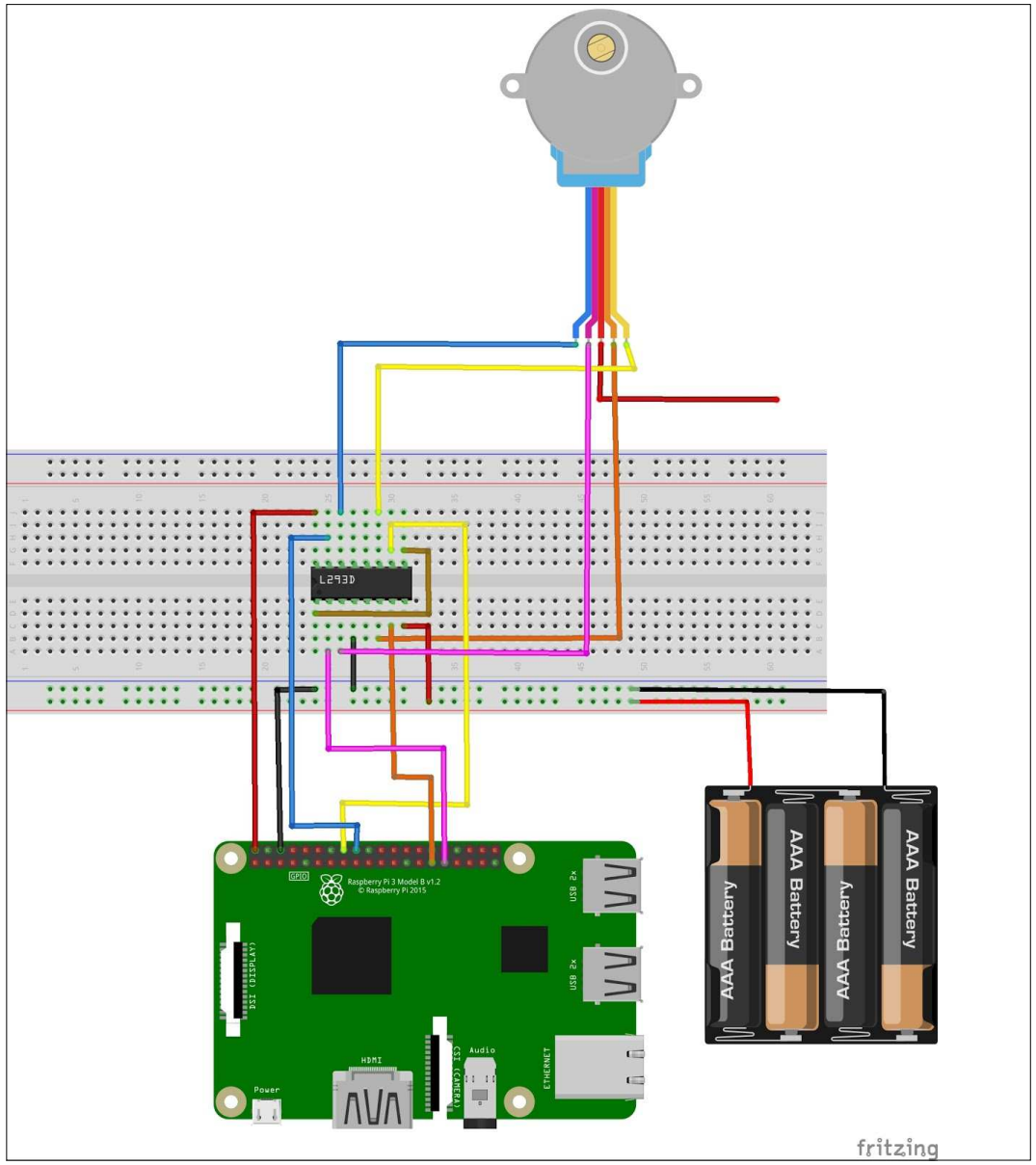


Les numéros montrés avec les couleurs des fils sont pour le 28BJY et les vôtres peuvent être différents. Le connecteur des fils (s'il y en a un) peut être différent d'une unité à l'autre. Vous pouvez utiliser un ohmmètre pour tester les bobines.

Le câblage

Un léger avertissement avant de commencer.

Tout d'abord, faites l'ensemble de votre câblage AVANT d'allumer le Raspberry Pi. Nous travaillons avec une source externe de puissance, donc faites attention de ne pas mettre en court-circuit certains fils ou d'alimenter la mauvaise broche.



Deuxièmement, VÉRIFIEZ votre câblage avant d'allumer votre RPi. Si vous vous êtes emmêlé, au mieux votre projet ne fonctionnera pas, et le moteur ne fera rien en buzzant.

Quand vous regardez le plan, ça semble plutôt simple (et ça l'est). Je me suis assuré que le câblage entre le Rpi et la puce de commande était de la même couleur que le segment correspondant du moteur. Nous n'allons utiliser que 4 fils sur les 5 du moteur. Le rouge (si le vôtre a un fil rouge) n'est pas connecté pour ce projet.

Puisque le composant principal dans ce projet est la puce de commande L293D, voici un petit plan pour essayer de vous faciliter la tâche :

```
L293D
Pin  1  -> Pin 9
Pin  2  -> Pi GPIO 6
Pin  3  -> Moteur rose
Pin  4  -> Rail négatif de la
           plaque
Pin  5  -> Non connecté
Pin  6  -> Moteur orange
Pin  7  -> Pi GPIO 5
Pin  8  -> Rail positif de la
           plaque
Pin  9  -> Pin 1
Pin 10 -> PI GPIO 23
Pin 11 -> Moteur jaune
Pin 12 -> Non connecté
Pin 13 -> Non connecté
Pin 14 -> Moteur bleu
Pin 15 -> Pi GPio 24
Pin 16 -> Pi +5VDC
```

Si vous respectez cela, vous ne devriez pas avoir de problème avec les fils.

Le code

Comme toujours, je vais parler du code par blocs. Allons-y.

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
```

```
GPIO.setwarnings(False)
coil_A_1_pin = 6 # rose
coil_A_2_pin = 5 # orange
coil_B_1_pin = 23 # bleu
coil_B_2_pin = 24 # jaune
```

Ici, nous ne définissons que les imports, nous paramétrons le mode GPIO (entrée/sortie universelle) et nous désactivons les alertes pour ne pas être ennuyé par des messages lorsque les broches sont déjà initialisées. Nous définissons aussi quelle broche universelle contrôle les bobines du moteur à travers la puce de commande.

```
# adaptez si c'est différent
StepCount = 8 Seq = range(0,
StepCount)
Seq[0] = [0,1,0,0]
Seq[1] = [0,1,0,1]
Seq[2] = [0,0,0,1]
Seq[3] = [1,0,0,1]
Seq[4] = [1,0,0,0]
Seq[5] = [1,0,1,0]
Seq[6] = [0,0,1,0]
Seq[7] = [0,1,1,0]
```

Et voici la clé permettant de faire fonctionner notre projet. Ce moteur veut 8 pas (internes) par tour de moteur (selon la doc.). Nous définissons également la séquence des bobines qui seront alimentées à chaque pas, sous la forme d'une série de listes. Chaque liste indique quelle bobine est alimentée à un moment donné.

```
GPIO.setup(coil_A_1_pin,
GPIO.OUT)
GPIO.setup(coil_A_2_pin,
GPIO.OUT)
GPIO.setup(coil_B_1_pin,
GPIO.OUT)
GPIO.setup(coil_B_2_pin,
GPIO.OUT)
```

Ici nous faisons le paramétrage, en déterminant quelles broches sont utilisées comme sorties.

```
def setStep(w1, w2, w3, w4):
GPIO.output(coil_A_1_pin, w1)
GPIO.output(coil_A_2_pin, w2)
GPIO.output(coil_B_1_pin, w3)
GPIO.output(coil_B_2_pin, w4)
```

Cette sous-routine est appelée chaque fois que nous voulons faire avancer le moteur d'un pas, et nous transmettons un 0 ou un 1 à chaque broche de fil de bobine sur la puce de commande pour alimenter ou pas les différentes bobines pour faire tourner le rotor.

```
def forward(delay, steps):
        for i in range(steps):
                for j in range(StepCount):
                        setStep(Seq[j][0], Seq[j][1], Seq[j][2], Seq[j][3])
                        time.sleep(delay)

def backwards(delay, steps):
        for i in range(steps):
                for j in reversed(range(StepCount)):
                        setStep(Seq[j][0], Seq[j][1], Seq[j][2], Seq[j][3])
                        time.sleep(delay)
```

Ces deux fonctions permettent, pour commander facilement le moteur en avant ou en arrière, un nombre spécifique de pas dans la direction appropriée.

```
if __name__ == '__main__':
        while True:
                delay = raw_input("Time Delay (ms)?")
                steps = raw_input("How many steps forward? ")
                forward(int(delay) / 1000.0, int(steps))
                steps = raw_input("How many steps backwards? ")
                backwards(int(delay) / 1000.0, int(steps))
```

Et enfin notre fonction « main » (principale) qui boucle sans cesse en demandant la quantité de temps et le nombre de pas dans cette direction donnée. Pour mon moteur, il faut 512 pas pour faire presque une rotation complète.

Sur mon système, avec mon moteur, un délai de 1 ms fonctionne bien. Mais vous devrez peut-être ajouter quelques millisecondes au vôtre pour qu'il fonctionne.

Notez que j'ai dit qu'il fallait 512 pas pour faire PRESQUE une rotation complète. Ce moteur a un rapport d'accélération de 64:1, pour lequel l'angle de pas est une fraction assez laide. Mais pour ce tutoriel, ça marche assez bien.

Si vous voulez en apprendre plus sur les moteurs pas-à-pas, adafruit.com propose un chouette petit article sur le sujet.

J'espère que vous avez aimé cette série jusqu'ici. Le prochain article vous apprendra à utiliser la carte microcontrôleur de l'Arduino. Nous utiliserons cette information dans la troisième partie de la série où nous contrôlerons l'Arduino avec un Raspberry Pi (ou tout autre ordinateur). Aussi, cela étant dit, pour la prochaine fois, vous devriez être prêt avec un Arduino (Uno ou Mega) et avoir sorti de leur carton les composants que nous avons utilisés au tout début de cette série.

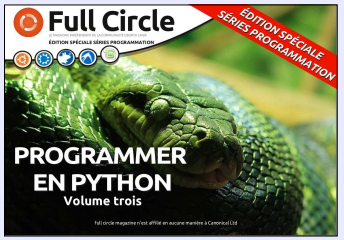
En attendant, continuez d'apprendre et surtout, AMUSEZ-VOUS !

Greg Walters est propriétaire de RainyDay Solutions LLC, une société de consultants à Aurora au Colorado, et programme depuis 1972. Il aime faire la cuisine, marcher, la musique et passer du temps avec sa famille.

Éditions Spéciales Python :

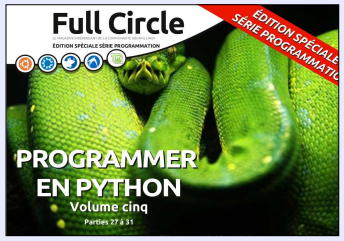
http://www.fullcirclemag.fr/?download/224

http://www.fullcirclemag.fr/?download/230



http://www.fullcirclemag.fr/?download/231

http://www.fullcirclemag.fr/?download/240



http://www.fullcirclemag.fr/?download/268

http://www.fullcirclemag.fr/?download/272

http://www.fullcirclemag.fr/?download/370

http://www.fullcirclemag.fr/?download/371

http://www.fullcirclemag.fr/?download/372

Écrire pour le Full Circle Magazine

Lignes directrices

Notre seule règle : tout article **doit avoir un quelconque rapport avec Ubuntu ou avec l'une de ses dérivées (Kubuntu, Xubuntu, Lubuntu, etc.).**

Autres règles

• Les articles ne sont pas limités en mots, mais il faut savoir que de longs articles peuvent paraître comme série dans plusieurs numéros.

• Pour des conseils, veuillez vous référer au guide officiel *Official Full Circle Style Guide* ici : http://url.fullcirclemagazine.org/75d471

• Utilisez n'importe quel logiciel de traitement de texte pour écrire votre article – je recommande LibreOffice –, mais le plus important est d'en **VÉRIFIER L'ORTHOGRAPHE ET LA GRAMMAIRE !**

• Dans l'article veuillez nous faire savoir l'emplacement souhaité pour une image spécifique en indiquant le nom de l'image dans un nouveau paragraphe ou en l'intégrant dans le document ODT (OpenOffice/LibreOffice).

• Les images doivent être en format JPG, de 800 pixels de large au maximum et d'un niveau de compression réduit.

• Ne pas utiliser des tableaux ou toute sorte de formatage en **gras** ou *italique*.

Lorsque vous êtes prêt à présenter l'article, envoyez-le par courriel à : articles@fullcirclemagazine.org.

Si vous écrivez une critique, veuillez suivre ces lignes directrices :

Traductions

Si vous aimeriez traduire le Full Circle dans votre langue maternelle, veuillez envoyer un courriel à ronnie@fullcirclemagazine.org et soit nous vous mettrons en contact avec une équipe existante, soit nous pourrons vous donner accès au texte brut que vous pourrez traduire. Lorsque vous aurez terminé un PDF, vous pourrez télécharger votre fichier vers le site principal du Full Circle.

Auteurs francophones

Si votre langue maternelle n'est pas l'anglais, mais le français, ne vous inquiétez pas. Bien que les articles soient encore trop longs et difficiles pour nous, l'équipe de traduction du FCM-fr vous propose de traduire vos « Questions » ou « Courriers » de la langue de Molière à celle de Shakespeare et de vous les renvoyer. Libre à vous de la/les faire parvenir à l'adresse mail *ad hoc* du Full Circle en « v.o. ». Si l'idée de participer à cette nouvelle expérience vous tente, envoyez votre question ou votre courriel à :

webmaster@fullcirclemag.fr

Écrire pour le FCM français

Si vous souhaitez contribuer au FCM, mais que vous ne pouvez pas écrire en anglais, faites-nous parvenir vos articles, ils seront publiés en français dans l'édition française du FCM.

CRITIQUES

Jeux/Applications

Si vous faites une critique de jeux ou d'applications, veuillez noter de façon claire :

• le titre du jeu ;
• qui l'a créé ;
• s'il est en téléchargement gratuit ou payant ;
• où l'obtenir (donner l'URL du téléchargement ou du site) ;
• s'il est natif sous Linux ou s'il utilise Wine ;
• une note sur cinq ;
• un résumé avec les bons et les mauvais points.

Matériel

Si vous faites une critique du matériel veuillez noter de façon claire :

• constructeur et modèle ;
• dans quelle catégorie vous le mettriez ;
• les quelques problèmes techniques éventuels que vous auriez rencontrés à l'utilisation ;
• s'il est facile de le faire fonctionner sous Linux ;
• si des pilotes Windows ont été nécessaires ;
• une note sur cinq ;
• un résumé avec les bons et les mauvais points.

Pas besoin d'être un expert pour écrire un article ; écrivez au sujet des jeux, des applications et du matériel que vous utilisez tous les jours.

MÉCÈNES

MÉCÈNES MENSUELS

2016:
Bill Berninghausen
Jack McMahon
Linda P
Remke Schuurmans
Norman Phillips
Tom Rausner
Charles Battersby
Tom Bell
Oscar Rivera
Alex Crabtree
Ray Spain
Richard Underwood
Charles Anderson
Ricardo Coalla
Chris Giltnane
William von Hagen
Mark Shuttleworth
Juan Ortiz
Joe Gulizia
Kevin Raulins
Doug Bruce
Pekka Niemi
Rob Fitzgerald
Brian M Murray
Roy Milner
Brian Bogdan
Scott Mack
Dennis Mack
John Helmers
JT
Elizabeth K. Joseph
Vincent Jobard
Chris Giltnane
Joao Cantinho Lopes
John Andrews

2017:

DONS UNIQUES

2016:
John Niendorf
Daniel Witzel
Douglas Brown
Donald Altman
Patrick Scango
Tony Wood
Paul Miller
Colin McCubbin
Randy Brinson
John Fromm
Graham Driver
Chris Burmajster
Steven McKee
Manuel Rey Garcia
Alejandro Carmona Ligeon
siniša vidović
Glenn Heaton
Louis W Adams Jr
Raul Thomas
Pascal Lemaitre
PONG Wai Hing
Denis Millar
Elio Crivello
Rene Hogan
Kevin Potter
Marcos Alvarez Costales
Raymond Mccarthy
Max Catterwell
Frank Dinger
Paul Weed
Jaideep Tibrewala
Patrick Martindale
Antonino Ruggiero
Andrew Taylor

2017:
Linda Prinsen
Shashank Sharma
Glenn Heaton
Frank Dinger

CHA CHA CHA CHANGEMENT

Notre administrateur est parti, pour de nombreux mois, sans rien dire à personne et je ne savais pas du tout, ni si, ni quand, les frais du site seraient ou ne seraient pas payés. Au départ, nous devions déménager le nom de domaine et le site, qui aurait été hébergé chez moi, et, finalement, j'ai réussi à retrouver l'admin et à me faire transférer le nom de domaine ainsi que l'hébergement du site.

Le nouveau site fonctionne dès à présent. D'ÉNORMES remerciements à Lucas Westermann (Monsieur Command & Conquer) d'avoir bien voulu prendre du temps sur ses loisirs pour recréer complètement le site, ainsi que les scripts, à partir de zéro.

J'ai fait la page Patreon pour pouvoir recevoir de l'aide financière pour ce qui concerne le domaine et les frais d'hébergement. L'objectif annuel a été atteint rapidement grâce à ceux dont les noms figurent sur cette page. Pas d'inquiétude à avoir : le FCM ne va pas disparaître. Plusieurs personnes ont demandé une option PayPal (pour un don ponctuel) et j'ai donc rajouté un bouton sur le côté du site.

Merci infiniment à tous ceux qui ont utilisé Patreon et le bouton PayPal. Cela m'a beaucoup aidé.

https://www.patreon.com/fullcirclemagazine

COMMENT CONTRIBUER

FULL CIRCLE A BESOIN DE VOUS !

Un magazine n'en est pas un sans articles et Full Circle n'échappe pas à cette règle. Nous avons besoin de vos opinions, de vos bureaux et de vos histoires. Nous avons aussi besoin de critiques (jeux, applications et matériels), de tutoriels (sur K/X/L/Ubuntu), de tout ce que vous pourriez vouloir communiquer aux autres utilisateurs de *buntu. Envoyez vos articles à :

articles@fullcirclemagazine.org

Nous sommes constamment à la recherche de nouveaux articles pour le Full Circle. Pour de l'aide et des conseils, veuillez consulter l'Official Full Circle Style Guide :
http://url.fullcirclemagazine.org/75d471

Envoyez vos **remarques** ou vos **expériences** sous Linux à : letters@fullcirclemagazine.org

Les tests de **matériels/logiciels** doivent être envoyés à : reviews@fullcirclemagazine.org

Envoyez vos **questions** pour la rubrique Q&R à : questions@fullcirclemagazine.org

et les **captures d'écran** pour « Mon bureau » à : misc@fullcirclemagazine.org

Si vous avez des questions, visitez notre forum : fullcirclemagazine.org

FCM n° 123

Date de parution du numéro en langue anglaise :
Vendredi 28 juillet 2017.

Équipe Full Circle

Rédacteur en chef - Ronnie Tucker
ronnie@fullcirclemagazine.org

Webmaster - Lucas Westermann
admin@fullcirclemagazine.org

Correction et Relecture
Mike Kennedy, Gord Campbell, Robert Orsino, Josh Hertel, Bert Jerred, Jim Dyer et Emily Gonyer

Remerciements à Canonical, aux nombreuses équipes de traduction dans le monde entier et à **Thorsten Wilms** pour le logo du FCM.

Pour la traduction française :
http://www.fullcirclemag.fr

Pour nous envoyer vos articles en français pour l'édition française :
webmaster@fullcirclemag.fr

Pour les Actus hebdomadaires du Full Circle :

Vous pouvez vous tenir au courant des Actus hebdomadaires en utilisant le flux RSS : http://fullcirclemagazine.org/feed/podcast

Ou, si vous êtes souvent en déplacement, vous pouvez obtenir les Actus hebdomadaires sur Stitcher Radio (Android/iOS/web) : http: www.stitcher.com/s?fid=85347&refid=stpr

et sur TuneIn à : http://tunein.com/radio/Full-Circle-Weekly-News-p855064/

Obtenir le Full Circle en français :

http://www.fullcirclemag.fr/?pages/Numéros

Obtenir le Full Circle Magazine :

Format EPUB - Les éditions récentes du Full Circle comportent un lien vers le fichier epub sur la page de téléchargements. Si vous avez des problèmes, vous pouvez envoyer un courriel à : mobile@fullcirclemagazine.org

Issuu - Vous avez la possibilité de lire le Full Circle en ligne via Issuu : http://issuu.com/fullcirclemagazine. N'hésitez surtout pas à partager et à noter le FCM, pour aider à le faire connaître ainsi qu' Ubuntu Linux.

Magzster - Vous pouvez aussi lire le Full Circle online via Magzster : http://www.magzter.com/publishers/Full-Circle. N'hésitez surtout pas à partager et à noter le FCM, pour aider à le faire connaître ainsi qu'Ubuntu Linux.