Raspberry Pi part n° 6

doper la framboise à coup de CANette

Tony Dixon (Royaume-Uni)

Si jusqu'à présent nous n'avions utilisé que les signaux numériques des interfaces UART, SPI et I²C, c'est que les interfaces analogiques brillent par leur absence sur le *Pi*. Pour ajouter un peu d'analogique, nous allons recourir à un convertisseur analogique-numérique (CAN) relié au bus SPI.

Une framboise qui a vraiment un pépin

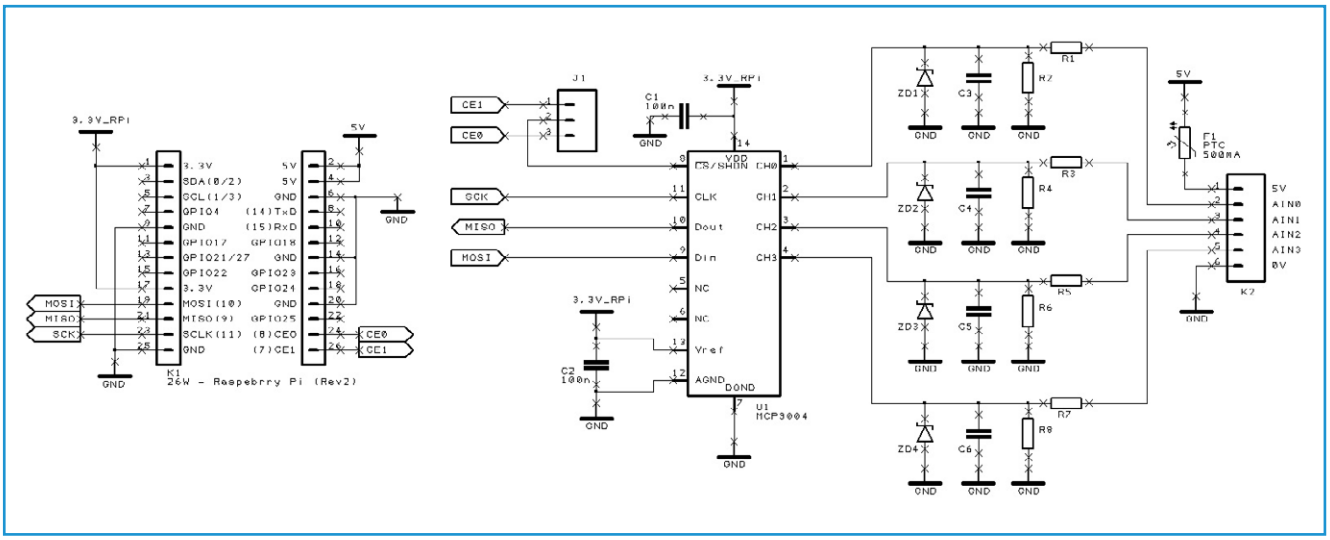
Le connecteur d'extension du *Pi* ne possède pas d'interface analogique. Pas une seule. Une honte quand on sait que des plateformes comme Arduino et *BeagleBone Black* en offrent plusieurs.
Le cas du *Pi* n'est heureusement pas désespéré puisqu'on peut toujours se servir d'un convertisseur A/N connecté à l'une de ses interfaces SPI ou I²C (ou aux deux).

L'interface SPI

Nous avons déjà parlé de l'interface SPI (*Serial Peripheral Interface*) dans l'Elektor. POST nº 9. Rappelons à ceux qui étaient partis aux framboises ce jour-là que l'interface SPI est accessible depuis les broches 19 (MOSI), 21 (MISO), et 23 (SCK), et que les deux signaux SCE (*SPI Chip Enable*) sont présents sur les broches 24 (CE0) et 26 (CE1) (**tableau 1**).
Voyez aussi l'encadré *Installation de la bibliothèque py-spidev* si vous n'avez ni installé le module Python pour SPI, ni configuré l'accès du *Pi* aux interfaces SPI.

Matériel CAN

Pour ce premier projet *Pi* analogique nous utiliserons le CAN MCP3004 de *Microchip* [1], une puce dotée de 4 canaux et d'une résolution de 10 bits.
La **figure 1** montre le schéma d'un MCP3004 relié à l'interface SPI du *Raspberry Pi*. Le cavalier J1 permet de choisir entre l'un des deux signaux *SPI chip enable*



(sélection de circuit), CE0 ou CE1. Le circuit est assez simple pour être assemblé sur une plaque standard.
Le MCP3004 est alimenté en 3,3 V, et nous devons faire attention aux tensions qui lui seront appliquées. Le circuit de conditionnement du signal est le même pour chaque canal du CAN. Pour le canal 1 p. ex., deux résistances R1 et R2 forment un diviseur de tension. Servons-nous de la formule du diviseur :

$$V_{out} = \frac{R2}{R1 + R2} \times V_{in}$$

- Pour des mesures de 0 V à 5 V, R1 = 10 kΩ et R2 = 10 kΩ donneront une lecture CAN comprise entre 0 V et 2,5 V.
- Pour des mesures de 0 V à 10 V, R1 = 10 kΩ et R2 = 22 kΩ donneront une lecture CAN comprise entre 0 V et 3,125 V.
- Pour des mesures de 0 V à 3,3 V, R1 = 10 kΩ et R2 absente donneront une lecture CAN comprise entre 0 V et 3,3 V.

Au besoin, nous pouvons choisir de la même façon les résistances des canaux 2, 3 et 4.

De même, p. ex. pour se débarrasser d'un certain bruit, nous pouvons filtrer le signal en choisissant pour les condensateurs C3 à C6 des valeurs de 1 nF à 10 nF.
Et pour protéger le circuit contre les surtensions, nous pouvons nous servir de zeners de 3,3 V pour les diodes ZD1 à ZD4.

Figure 1. Schéma de principe du CAN MCP3004 relié au *Pi*.

Tableau 1. Brochage du connecteur d'extension

nom de la broche	fonction	autre fonction	RPi.GPIO
P1-02	5,0V	-	-
P1-04	5,0V	-	-
P1-06	GND	-	-
P1-08	GPIO14	UART0_TXD	RPi.GPIO8
P1-10	GPIO15	UART0_RXD	RPi.GPIO10
P1-12	GPIO18	PWM0	RPi.GPIO12
P1-14	GND	-	-
P1-16	GPIO23		RPi.GPIO16
P1-18	GPIO24		RPi.GPIO18
P1-20	GND	-	-
P1-22	GPIO25		RPi.GPIO22
P1-24	GPIO8	**SPI0_CE0_N**	RPi.GPIO24
P1-26	GPIO7	**SPI0_CE1_N**	RPi.GPIO26

nom de la broche	révision 1 de la carte		révision 2 de la carte	
	fonction	autre fonction	fonction	autre fonction
P1-01	3,3V	-	3,3V	-
P1-03	GPIO0	I2C0_SDA	GPIO2	I2C1_SDA
P1-05	GPIO1	I2C0_SCL	GPIO3	I2C1_SCL
P1-07	GPIO4	GPCLK0	GPIO4	GPCLK0
P1-09	GND	-	GND	-
P1-11	GPIO17	RTS0	GPIO17	RTS0
P1-13	GPIO21		GPIO27	
P1-15	GPIO22		GPIO22	
P1-17	3,3V	-	3,3V	-
P1-19	GPIO10	**SPI0_MOSI**	GPIO10	**SPI0_MOSI**
P1-21	GPIO9	**SPI0_MISO**	GPIO9	**SPI0_MISO**
P1-23	GPIO11	**SPI0_SCLK**	GPIO11	**SPI0_SCLK**
P1-25	GND	-	GND	-

Note : I2C0_SDA et I2C0_SCL (GPIO0 & GPIO1), ainsi que I2C1_SDA et I2C1_SCL (GPIO2 & GPIO3) sont dotées de résistances de rappel de 1,8 kΩ au 3,3 V.

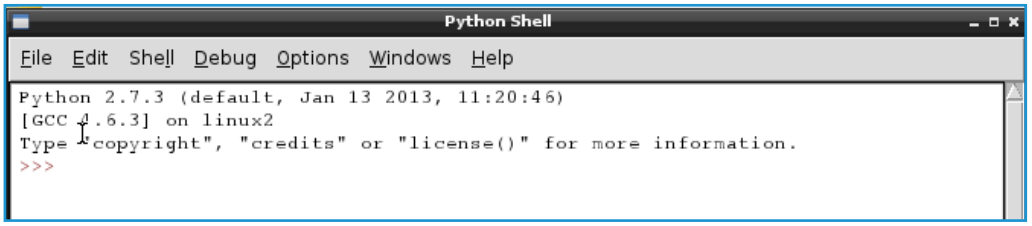


Figure 2.
L'environnement IDLE pour Python.

Programme d'exemple : mcp3004.py

Le circuit étant assemblé et *spidev* installé (voir l'encadré *Installation de la bibliothèque py-spidev*), écrivons un court programme de test qui mesurera et affichera la tension du canal 1 du CAN.

Double-cliquez sur l'icône IDLE du bureau pour lancer l'EDI et la console Python (**fig. 2**).

Dans le menu *File*, sélectionnez *New Win-*

Installation de la bibliothèque py-spidev

Nous avons déjà utilisé le module Python pour SPI appelé *py-spidev* lors du projet nº 9. Pour ceux qui auraient manqué la procédure d'installation, entrez les commandes suivantes dans une console *Lxterminal* (**fig. 4**) :

```
sudo apt-get install git-core

cd ~
git clone git://github.com/doceme/py-spidev
cd py-spidev/
sudo python setup.py install
```

Vous pouvez aussi utiliser l'installeur de paquets Python *pip* :

```
sudo apt-get install git-core python-dev
sudo apt-get install python-pip
sudo pip install spidev
```

L'interface SPI matérielle est désactivée par défaut, vous devez l'activer en modifiant le fichier *blacklist*. Ouvrez-le :

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

Trouvez la ligne qui contient **blacklist spi-bcm2708**, commentez-la en ajoutant le caractère # au début de la ligne, puis sauvegardez le fichier. *Pi* doit être redémarré pour que le changement soit pris en compte :

```
sudo reboot
```

Lancez une nouvelle session LXTerminal et, pour vérifier que vous avez bien deux fichiers de périphériques SPI (un pour chaque signal *SPI Chip Select*), entrez :

```
ls /dev/spi*
```

La sortie de cette commande devrait être :

```
/dev/spidev0.0
/dev/spidev0.1
```

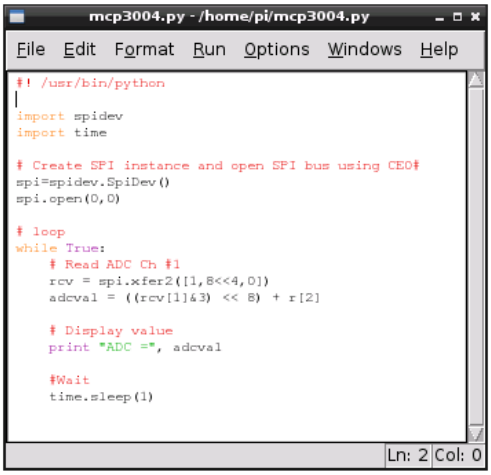


Figure 3. L'éditeur de IDLE.

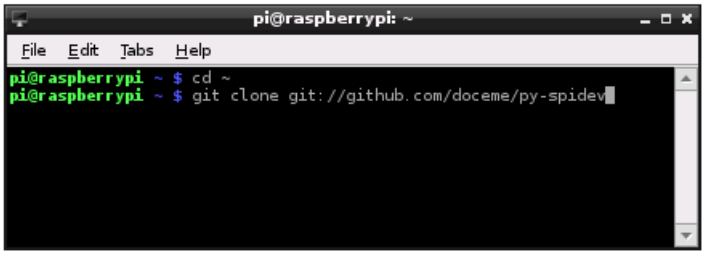


Figure 4. LXTerminal.

dow pour créer un nouveau programme dans l'éditeur. Copiez-y le code du **listage 1** (**fig. 3**).
Sauvegardez le programme saisi. Retournez ensuite dans la console *Lxterminal*, et entrez la commande suivante pour le rendre exécutable :

```
chmod +x mcp3004.py
```

Et pour exécuter ce programme :

```
sudo ./mcp3004.py
```

(130260 – version française : Hervé Moreau)

Listage 1

```
#! /usr/bin/python

import spidev
import time

# Create SPI instance and open SPI bus using CE0
spi = spidev.SpiDev()
spi.open(0,0)

# Loop
while True:

    # Read ADC Ch #1
    rcv = spi.xfer2 ([1,8<<4, 0])
    adcval = ((rcv [1]&3) << 8) + rcv[2]

    # Display value
    Print "ADC = ", adcval

    # Wait
    time.sleep(1)
```

Liens

[1] http://goo.gl/eMSOQ

[2] https://github.com/doceme/py-spidev

Les commandes spidev	
spi.open (0,0)	ouvre le bus SPI 0 en utilisant CE0
spi.open (0,1)	ouvre le bus SPI 0 en utilisant CE1
spi.close ()	déconnecte l'objet de l'interface
spi.writebytes ([array of bytes])	écrit un tableau d'octets dans le périphérique SPI
spi.readbytes (len)	lit *len* octets reçus du périphérique SPI
spi.xfer2 ([array of bytes])	envoie un tableau d'octets en laissant le signal CEx actif en permanence
spi.xfer ([array of bytes])	envoie un tableau d'octets en désactivant et réactivant le signal CEx à chaque octet transmis