Raspberry Pi Sensors

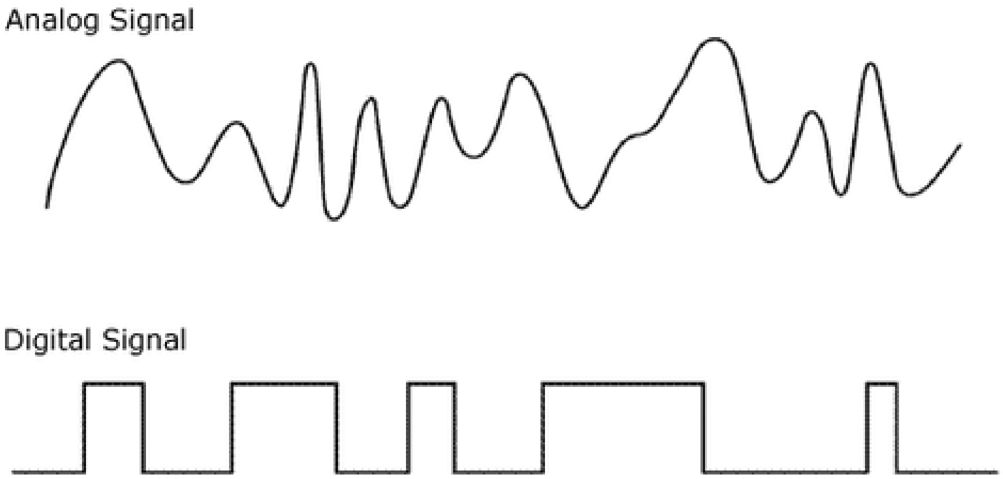
Author: Nick Lee

We are going to hook up a number of sensors to enable Raspberry Pi to detect its environment.

Raspberry Pi is a digital computer, and a digital computer only knows two numbers: 1 and 0.

A lot of physical quantities, e.g. temperature, humidity, wind speed, etc., are smooth-varying numbers. They can take on values such as 25.6, 98.76, or 33.3333333333. These are called analog quantities.

For Raspberry Pi to understand analog quantities, a sensor has to convert those analog quantities to digital signals.

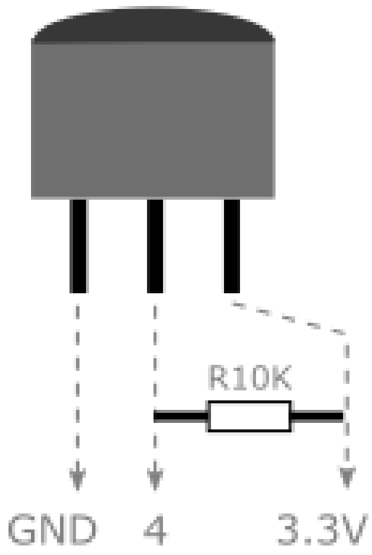


However, there are many digital "languages". Widely used are 1-wire, I2C, and SPI. A sensor may speak any of these, and you have to configure Raspberry Pi to speak the sensor's "language" before they can communicate.

DS18B20

temperature sensor

First, we try a temperature sensor. Wire it up according to the diagram below. You will need a 10K pull-up resistor.



It speaks the 1-wire protocol. We have to enable 1-wire on Raspberry Pi.

Enable 1-wire

```
$ sudo nano /boot/config.txt

dtoverlay=w1-gpio

$ sudo reboot
```

Add this line. It causes Linux to load the 1-wire modules.

```
$ lsmod | grep w1
```

Check whether the 1-wire modules are loaded. If not, check previous steps.

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

What do you see?

Use the sensor package

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Install software for I2C:

```
$ sudo apt-get install i2c-tools python3-smbus
```

Install software for SPI:

```
$ sudo apt-get install python3-dev
$ sudo pip3 install spidev
```

Finally, install the sensor package:

```
$ sudo pip3 install sensor
```

```
$ python3
```

```
>>> from sensor.DS18B20 import DS18B20
>>> ds = DS18B20('28-ZZZZZZZZ')
>>> ds.temperature()
```

What do you see? How do you access different units of the temperature?

```
>>> t = ds.temperature()
>>> t.C
28.937
>>> t.F
84.0866
```

The **sensor** package supports all sensors we are going to use in this course.

Next, we try to display the temperature on an LCD, which speaks the **I2C** protocol. We have to enable I2C on Raspberry Pi.

Enable I2C

```
$ sudo nano /etc/modules

i2c-dev
```

Add this line

```
$ sudo nano /boot/config.txt

dtparam=i2c_arm=on
```

Add this line. It causes Linux to load the I2C modules.

```
$ sudo reboot
```

```
$ lsmod | grep i2c
```

Check whether the I2C modules are loaded. If not, check previous steps.

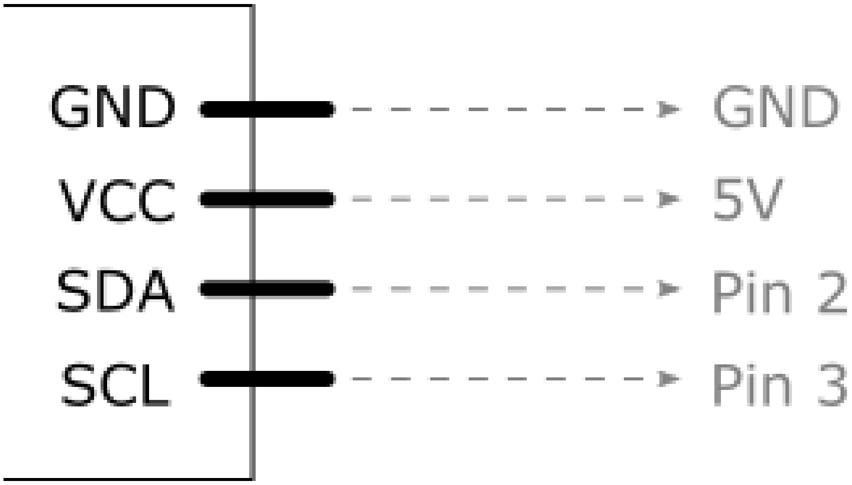
```
$ i2cdetect -y 1

     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Nothing is on the I2C bus, because nothing has been attached yet.

LCD1602

Wire it up:



All I2C devices share 2 wires: SDA (data) and SCL (clock). Their wiring diagrams are nearly identical, except that most sensors use 3.3V power supply instead of 5V. Read their datasheets.

```
$ i2cdetect -y 1
```

Do you see the **address** of the LCD?

Display on the LCD

```
$ python3

>>> from sensor.LCD1602 import LCD1602

>>> lcd = LCD1602(1, ??address??)

>>> lcd.display('Nick Lee', 1)
>>> lcd.display('Hong Kong', 2)

>>> lcd.clear()
```

Can you write a program that displays the temperature on the LCD?

display.py

```
import time
from sensor.DS18B20 import DS18B20
from sensor.LCD1602 import LCD1602

ds = DS18B20('??address??')
lcd = LCD1602(1, ??address??)

try:
    while 1:
        t = ds.temperature()
        lcd.display('%.1f C' % t.C, 1) ❶

        time.sleep(1)
finally: ❷
    lcd.clear()
```

❶ Display Celsius to 1 decimal place

❷ When you Ctrl-C to kill the infinite loop, an exception is raised and the **`finally`** clause is executed, clearing the display.

HTU21D

humidity and temperature sensor

Wiring: nearly identical to LCD1602, except that supply voltage is 3.3V.

```
$ i2cdetect -y 1
```

What is the sensor's address?

```
$ python3

>>> from sensor.HTU21D import HTU21D

>>> htu = HTU21D(1, ??address??)

>>> h = htu.humidity()
>>> h.RH

>>> t = htu.temperature()
>>> t.C
```

BMP180

pressure and temperature sensor

Wiring: nearly identical to LCD1602, except that supply voltage is 3.3V.

```
$ i2cdetect -y 1
```

What is the sensor's address?

```
$ python3

>>> from sensor.BMP180 import BMP180

>>> bmp = BMP180(1, ??address??)

>>> p = bmp.pressure()
>>> p.hPa

>>> t = bmp.temperature()
>>> t.C


# Look up mean sea level pressure from local observatory.
# 1009.1 hPa is only for example.
>>> a = p.altitude(msl=1009.1)
>>> a.m
```

Can you write a program that displays the temperature, humidity, and pressure on the LCD?

display.py

```
import time
from sensor.DS18B20 import DS18B20
from sensor.HTU21D import HTU21D
from sensor.BMP180 import BMP180
from sensor.LCD1602 import LCD1602

ds = DS18B20('??address??')
htu = HTU21D(1, ??address??)
bmp = BMP180(1, ??address??)
lcd = LCD1602(1, ??address??)

try:
    while 1:
        t = ds.temperature()
        h = htu.humidity()
        p = bmp.pressure()

        ln1 = '%.1f C    %.1f%%' % (t.C, h.RH) ❶
        ln2 = '   %.1f hPa' % p.hPa ❷

        lcd.display(ln1, 1)
        lcd.display(ln2, 2)

        time.sleep(1)
finally:
    lcd.clear()
```

❶ On first line, display Celsius and Relative Humidity.

❷ On second line, display Hectopascal.

Run script on startup

We are going to create a **systemd** service. Suppose we call the service **indoormonitor:**

```
$ cd /lib/systemd/system
$ sudo nano indoormonitor.service
```

```
[Unit]
Description=Indoor Monitoring Service
After=network.target

[Service]
ExecStart=/usr/bin/python3 /path/to/script.py arguments ...

[Install]
WantedBy=multi-user.target
```

```
$ sudo systemctl enable indoormonitor.service
$ sudo systemctl start indoormonitor.service
```

You may always enquire about the service's status with:

```
$ sudo systemctl status indoormonitor.service
```

Deal with Analog Signals

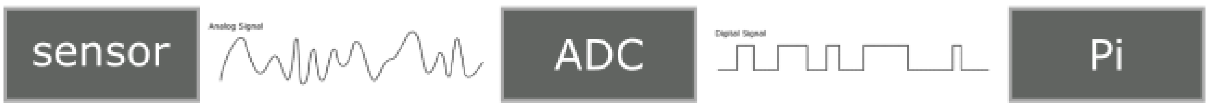
All sensors we have used so far convert analog quantities to a digital format, so Raspberry Pi can read the numbers readily. **What if we got a sensor that speaks analog signals?**

For example, a **TMP36** temperature sensor outputs a varying voltage proportional to the temperature measured.

Another example is a **photoresistor**, whose resistance changes with light intensity.

How could Raspberry Pi read these non-digital values?

The solution is to insert an **analog-to-digital converter (ADC)** between the sensor and the Pi.



For this course, we are going to detect light intensity using a photoresistor coupled with an ADC chip, either MCP3004 or MCP3008.

Enable SPI

MCP3004/MCP3008 speaks SPI with Raspberry Pi.

```
$ sudo nano /boot/config.txt

dtparam=spi=on
```

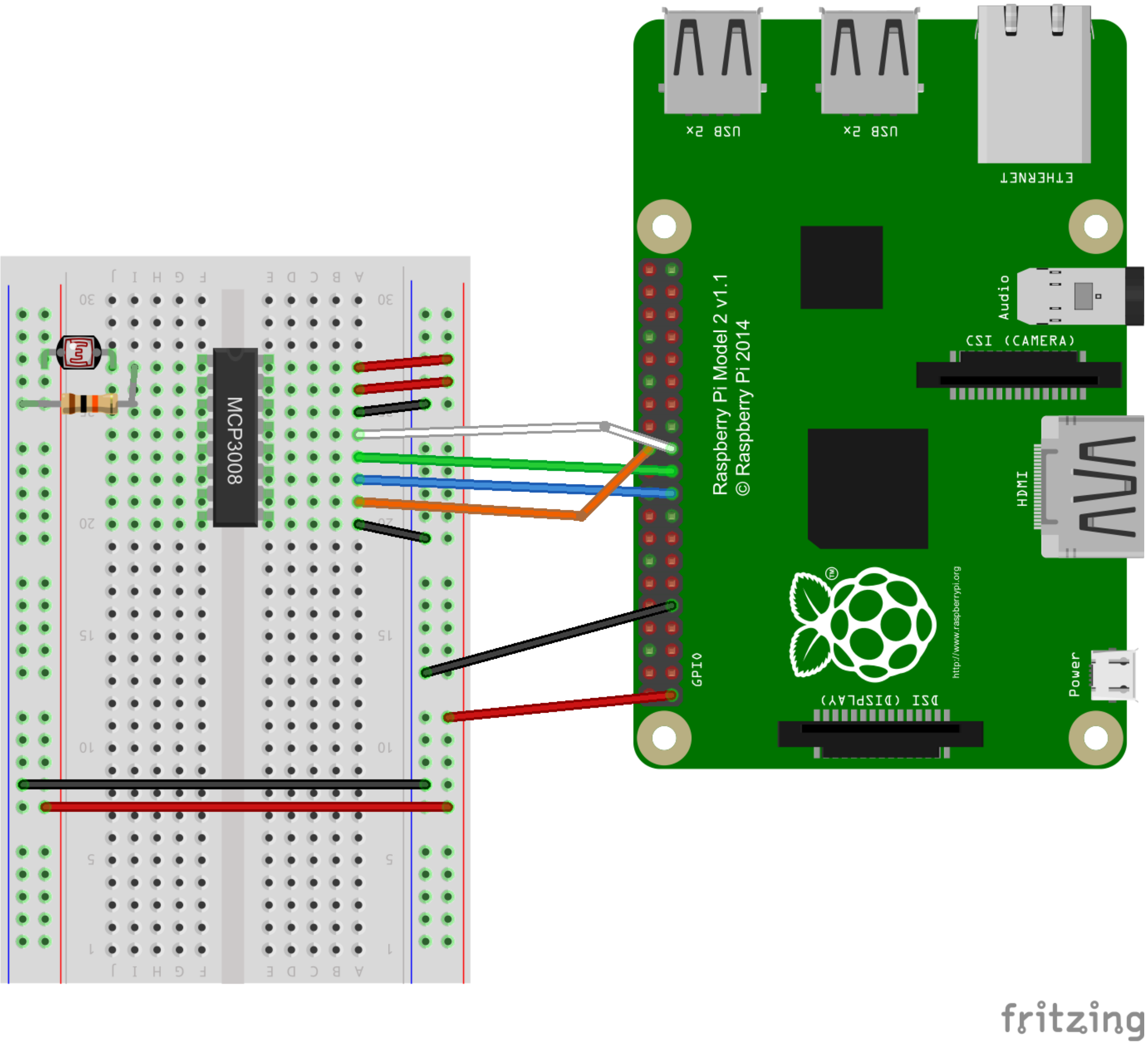
Add this line. It causes Linux to load the SPI modules.

```
$ sudo reboot
```

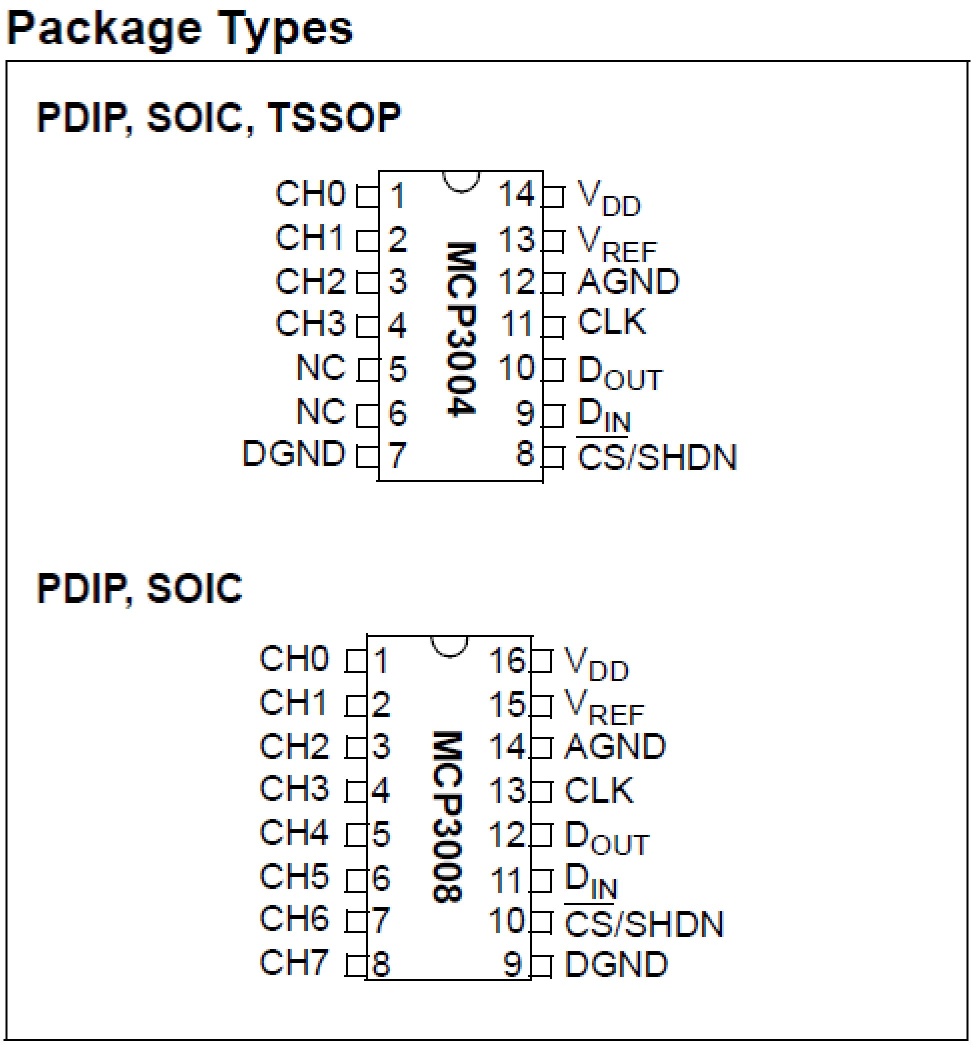
```
$ lsmod | grep spi
```

Check whether the SPI modules are loaded. If not, check previous steps.

Photoresistor and MCP3008



- the ADC uses a chip select of 0
- the reference voltage is 3.3V
- the photoresistor and 10K resistor form a voltage divider, and the result is tapped by channel 0
- as the photoresistor's resistance changes, channel 0's voltage also changes.

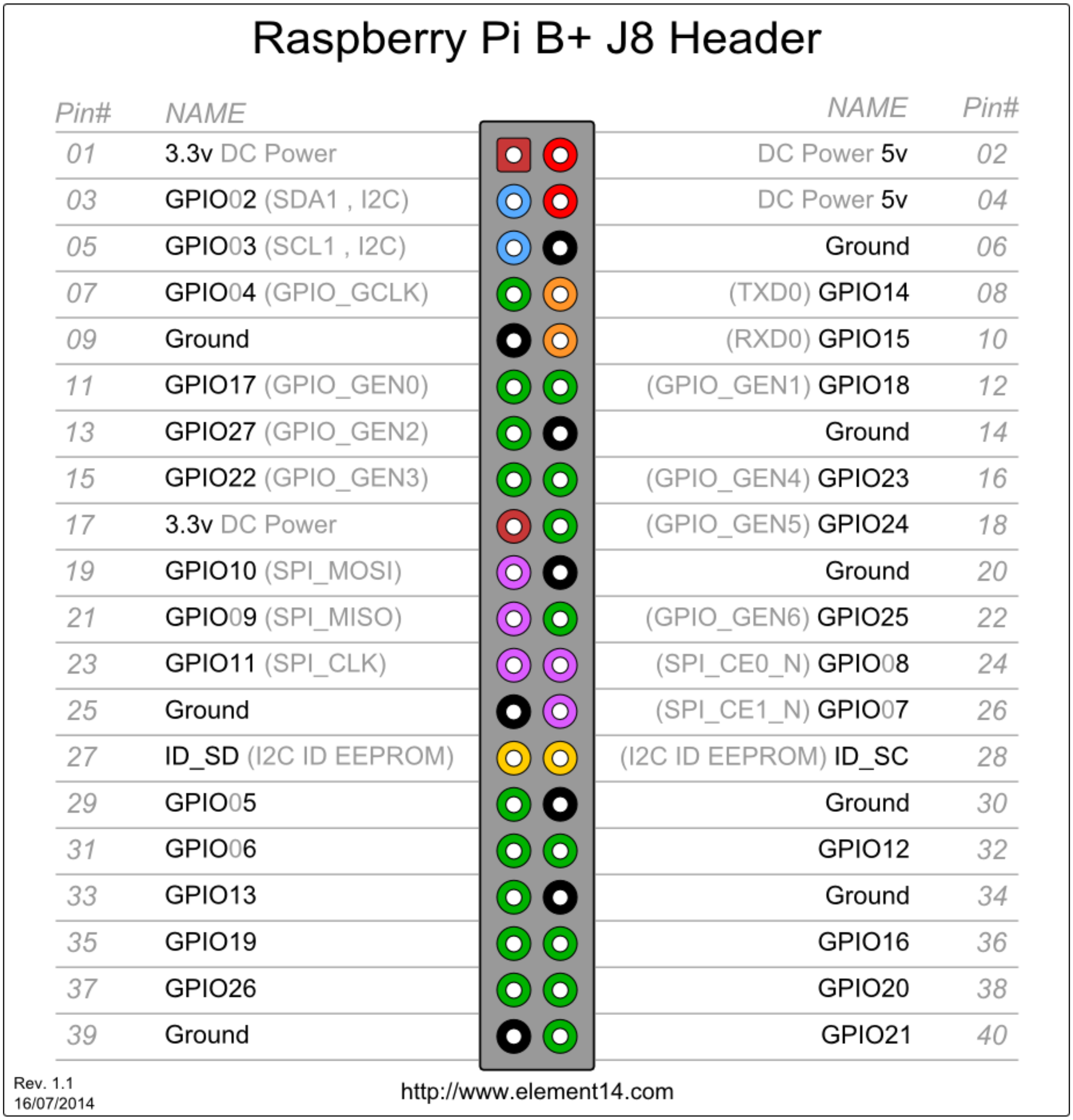


The pinouts of MCP3004 and MCP3008 are very similar. The sensor package, while explicitly supporting MCP3004, can be used to read MCP3008 as well.

```
$ python3

>>> from sensor.MCP3004 import MCP3004
>>> mcp = MCP3004(bus=0, addr=0, vref=3.3)
>>> mcp.voltage(0)   # read channel 0
```

Varying the light intensity, you should notice the voltage changes too.



Raspberry Pi B+ J8 Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I2C)	DC Power 5v	04
05	GPIO03 (SCL1 , I2C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	(I2C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 1.1
16/07/2014

http://www.element14.com