Programmation Hardware

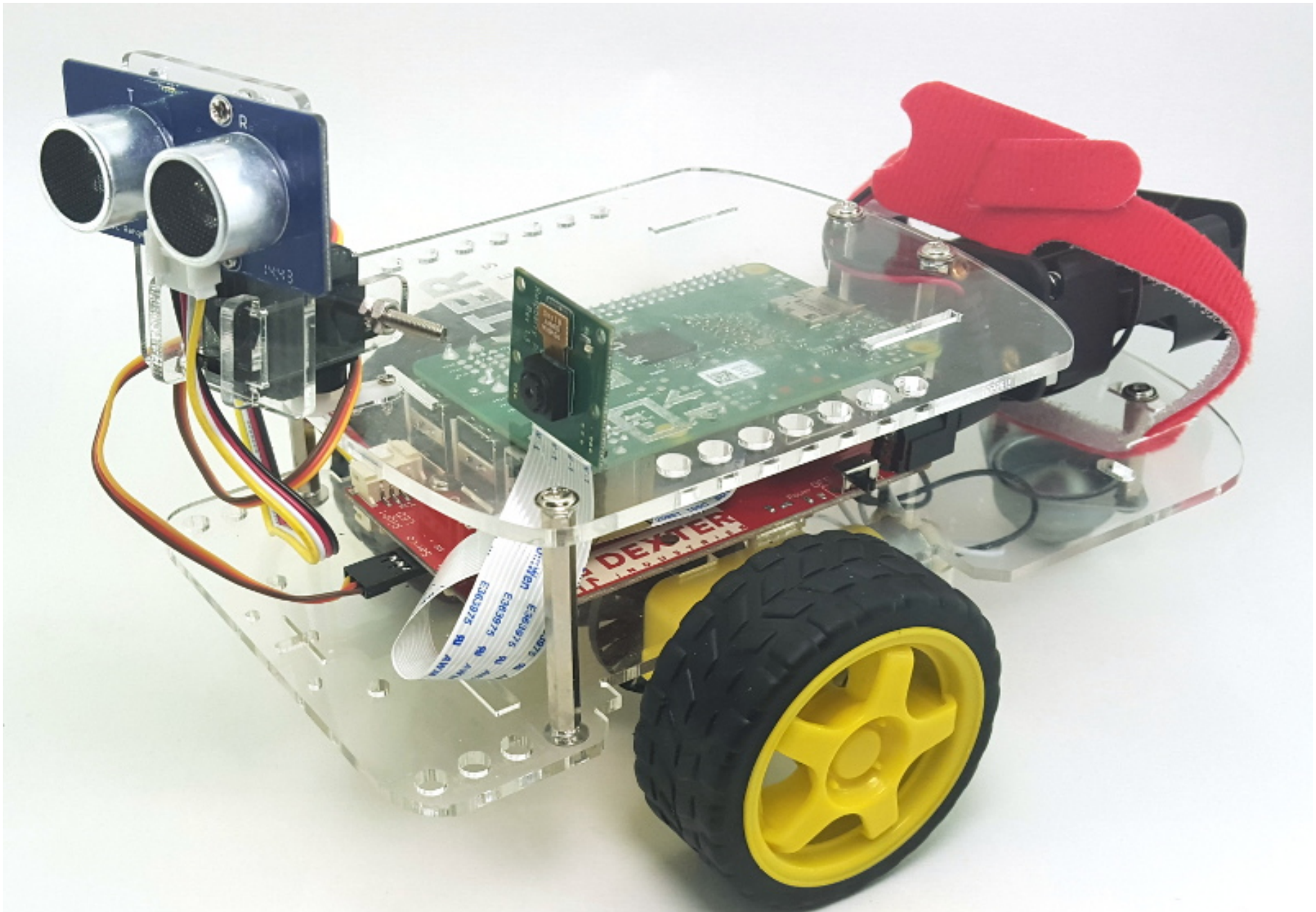
Nicolas Baskiotis
nicolas.baskiotis@sorbonne-universite.fr

équipe MLIA, Institut des Systèmes Intelligents et de Robotique
Sorbonne Université

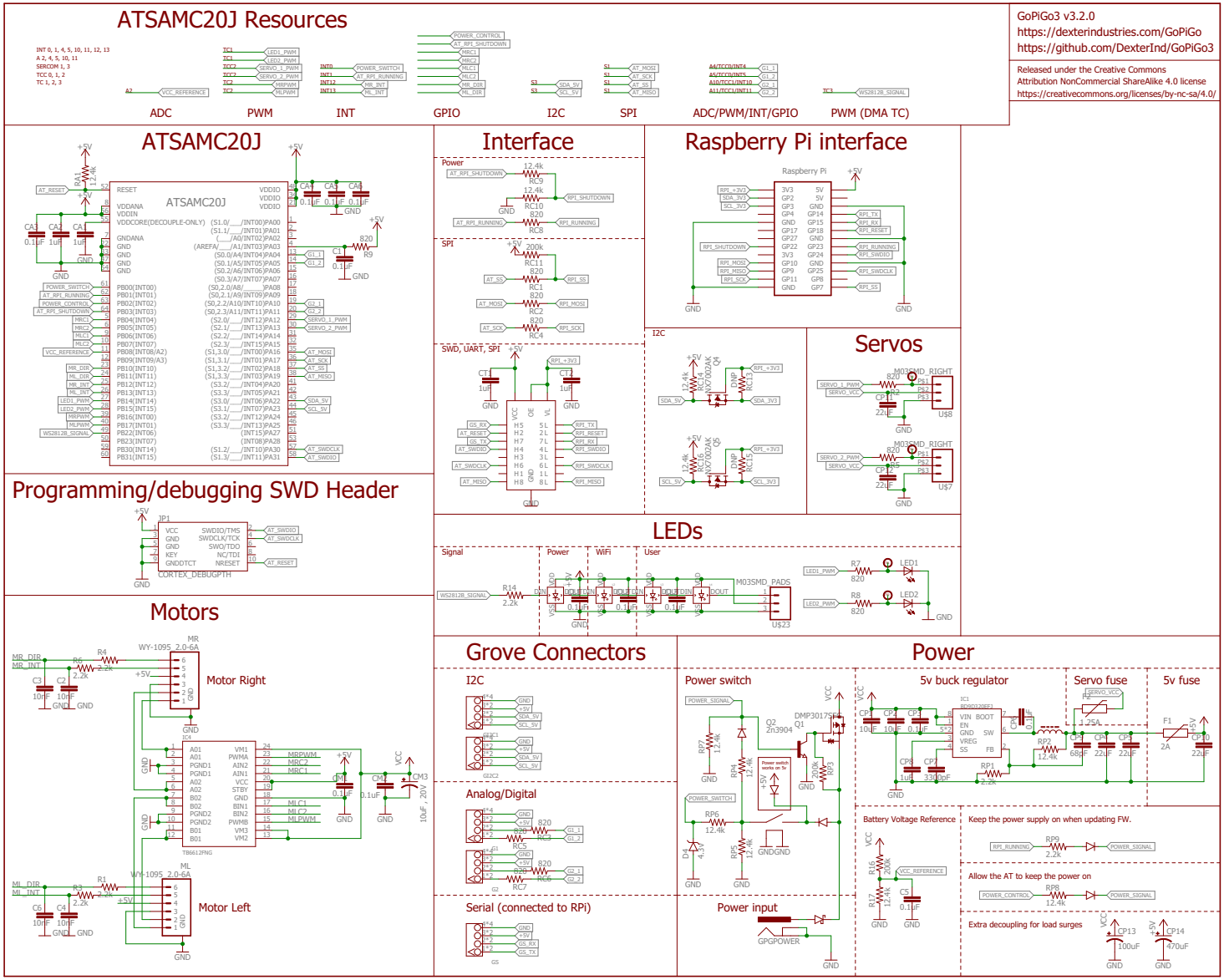
S2 (2022-2023)

Plan

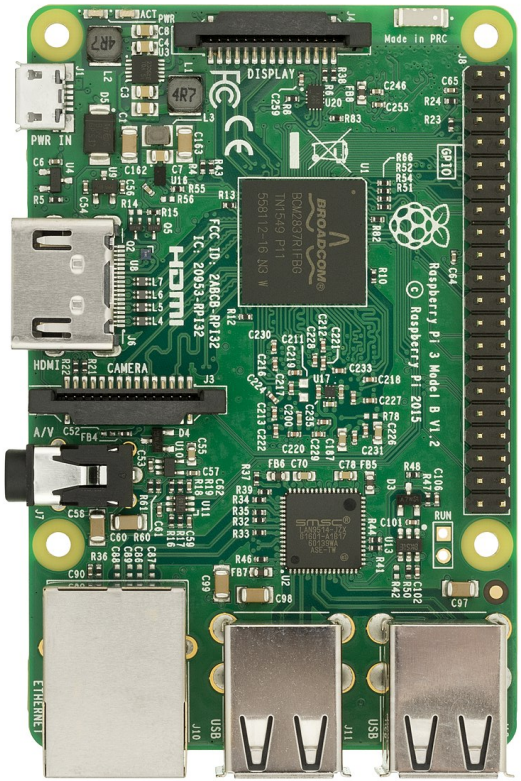
Vu d'ensemble



Vu d'ensemble



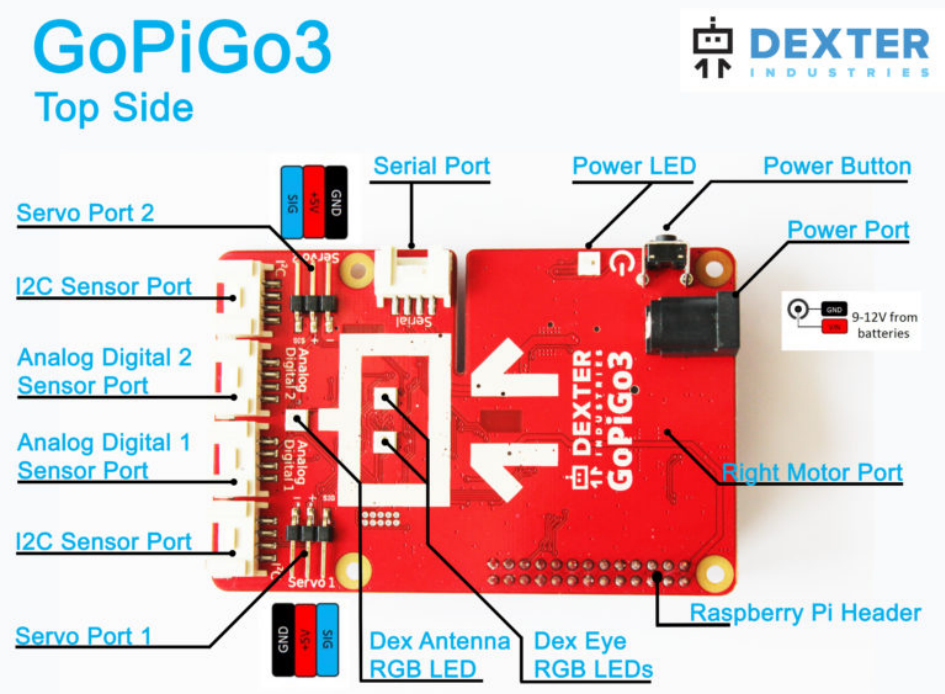
Raspberry Pi



RPi 3 - Model B Technical Specification

- Broadcom BCM2387 chipset
- 1.2GHz Quad-Core ARM Cortex-A53, 1GB RAM, 64 Bit CPU
- 802.11 bgn Wireless LAN, Bluetooth 4.1
- 4 x USB ports
- Full size HDMI
- 10/100 BaseT Ethernet socketbr
- camera port for connecting the Raspberry Pi camera
- Micro SD port
- 40pin extended GPIO

Micro-controlleur Dexter



Specifications

- Communication with the GoPiGo3 board occurs over the SPI interface.
- 2x I2C Sensor Ports
- 2x Analog Digital Sensor Ports (Analog, digital, and I2C Grove devices)
- Serial Port
- 2x Servo Connections, controlled by the micro-controller on the GoPiGo3.
- 2 Magnetic encoders

Plan

Bus de communication

Transférer des données : il faut pouvoir

- émettre des données
- recevoir des données
- choisir vers quoi/de quoi
- avec pour seul moyen un courant continu !

$\Rightarrow$ le **bus de communication** informatique :

- système de transfert de données par l'intermédiaire d'une voie de transmission commune
- les composants sur un même bus ne prennent pas part à la transmission des données des autres composants

Vocabulaire

- **Adresse** : identification de chaque composant
- **Contrôle** : signaux permettant d'identifier le type d'action (écriture, lecture, taille des messages)
- **Données** : le message en lui-même

GPIO : General Purpose Input/Output

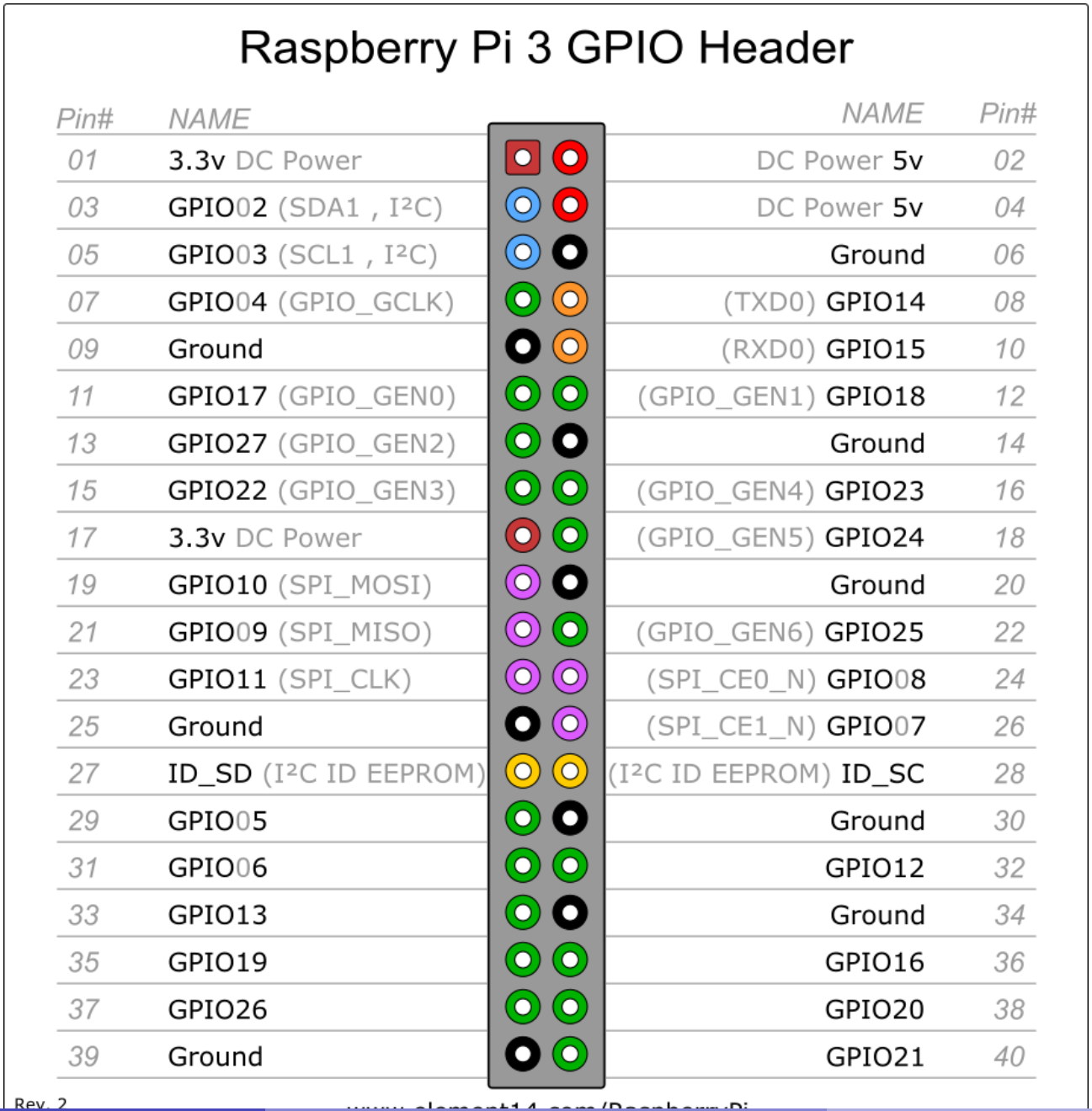
Le protocole binaire

- 1 seul pin
- Deux états : **HIGH** ou **LOW** (courant ou pas de courant)
- Peut transférer des informations ou des données simples (allumer/éteindre une LED, détecter un événement, . . .)

Exemple

```
import RPi.GPIO as GPIO # RPi: module Raspberry Pi de Raspbian (linux pour
GPIO.setmode(GPIO.BOARD) # ou GPIO.setmode(GPIO.BCM), type de numerotation
GPIO.setup(12, GPIO.IN) #configure le pin 12 en lecture
GPIO.setup(13, GPIO.OUT) #configure le pin 13 en ecriture
# setup obligatoire, configure pull-up resistance
state = GPIO.input(12) # lecture, 0/1 ou GPIO.HIGH/LOW
GPIO.output(13, GPIO.HIGH) # ecriture
GPIO.cleanup() # nettoyage
GPIO.wait_for_edge(12,GPIO.RISING) #attend un signal [FALLING,BOTH]
GPIO.add_event_detect(12,GPIO.FALLING,callback=myfun) # asynchrone
if GPIO.event_detected(12): ...
```

Raspberry GPIO



Du digital à l'analogique

Problème : le RPi n'a pas de port analogique

- Digital : valeurs discrètes finies (binaire)
- Analogique : valeurs continues

$\Rightarrow$ Comment passer de l'information analogique ?

Du digital à l'analogique

Problème : le RPi n'a pas de port analogique

- Digital : valeurs discrètes finies (binaire)
- Analogique : valeurs continues

$\Rightarrow$ Comment passer de l'information analogique ?

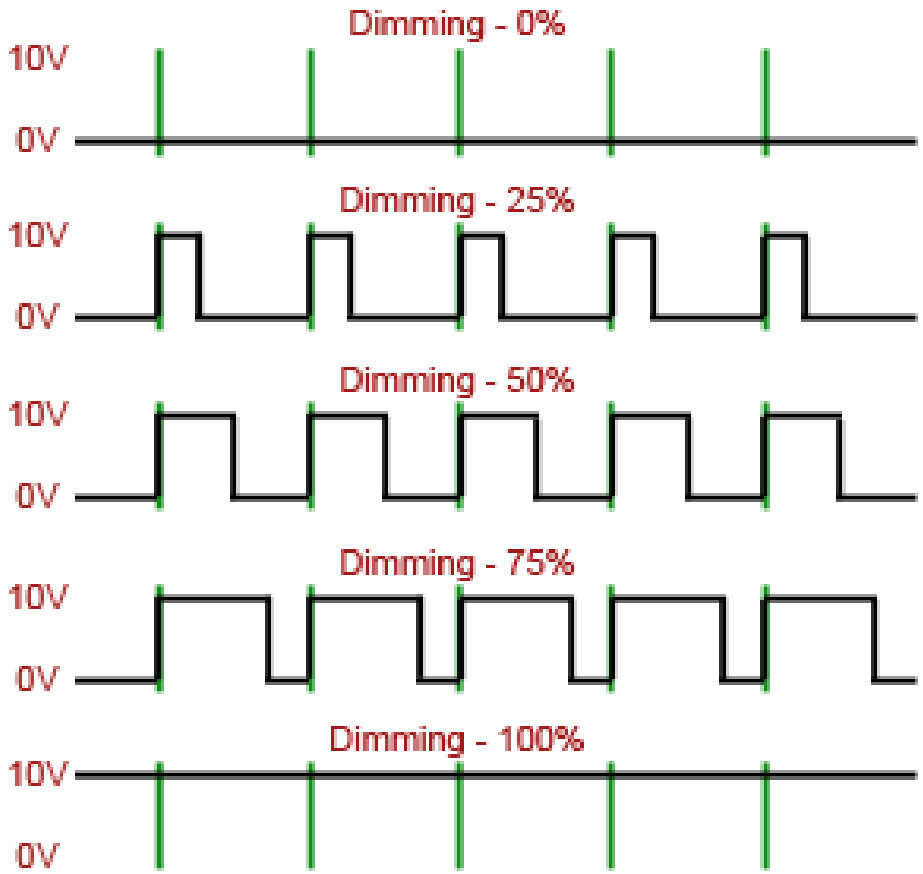
Encoder l'information

- Convertir les valeurs continues en nombres discrets
  $\rightarrow$ compliqué (quand fini un bit ? quand commence un autre ?)
- Convertir en un rapport de 0 à 100%
  $\rightarrow$ Codage fréquentiel

Pulse Width Modulation (PWM)

Principe

- Codage par signal en créneau
- A une fréquence donnée (sans importance)
- encodage par le % à **HIGH** du signal



A la main :

```
while True:
    GPIO.output(pin,HIGH)
    time.sleep(0.8)
    GPIO.output(pin,LOW)
    time.sleep(0.2)
```

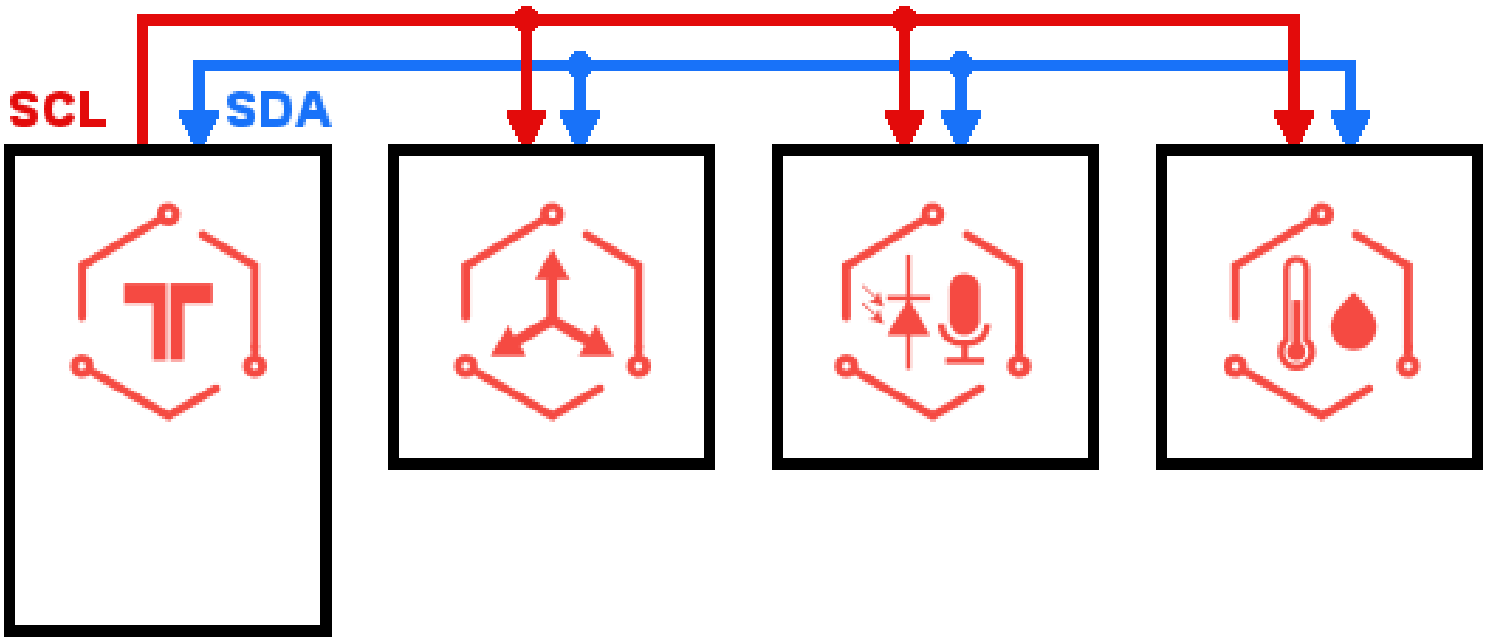
Paquet GPIO :

```
p = GPIO.PWM(channel, frequency)
p.ChangeDutyCycle(50.0)#entre 0 et 100.0
p.stop()
```

Bus I2C : Inter-Integrated Circuit

Spécifications

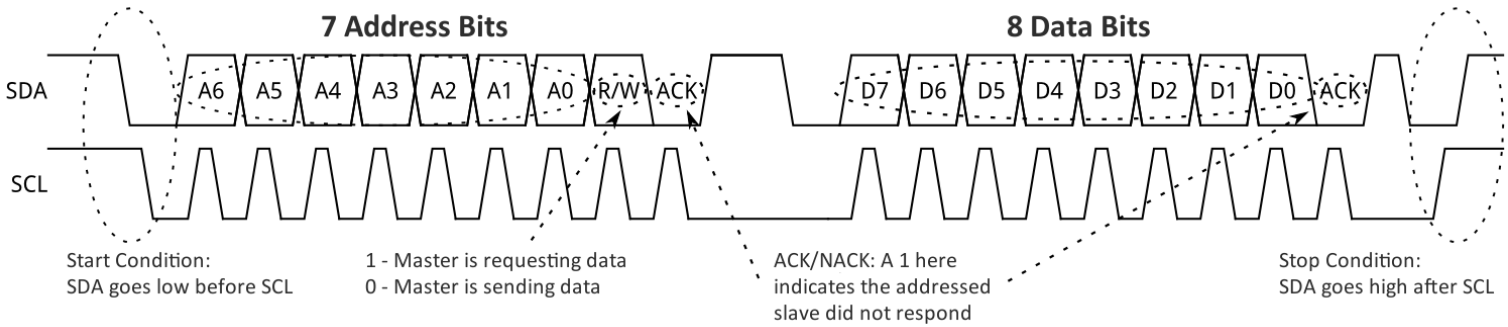
- plusieurs devices sur un simple bus, bi-directionnels
- Protocole Master/Slave : un maître décide qui parle/à qui il parle.
- utilise 2 pins (2 lignes)
  - Besoin de synchronisation : à quel rythme s'échange 1 bit ? ⇒ Clock signal (signal d'horloge, SCL)
  - Une ligne de communication, d'échanges de données (SDA)



Bus I2C : Inter-Integrated Circuit

Fonctionnement

- Avant chaque bit échangé sur SDA, le bit SCL est allumé : il indique que la prochaine valeur est disponible.
- 1 bloc d'adresse de transmis sur 7 bit, le dernier indique lecture/écriture
- les données suivent ensuite.



Principalement pour des senseurs simples : lecture de température, de distance, . . .
Problème : communication dans un seul sens, données peu complexes, plutôt lent.

Bus I2C : Inter-Integrated Circuit

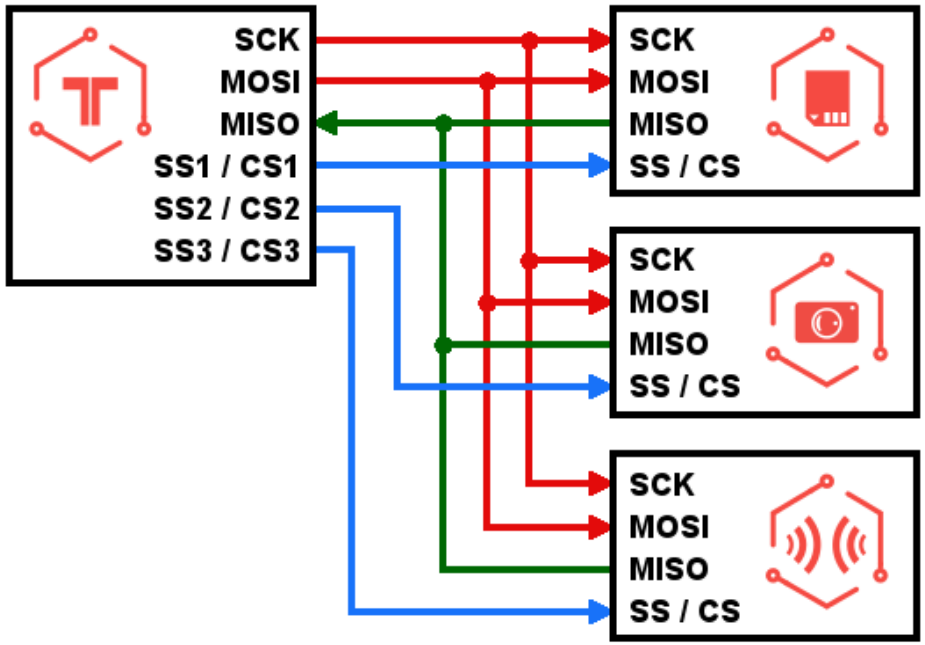
En python : paquet `smbus`

```
from smbus import SMBus
bus = SMBus(0) # numero du bus
#lire du device a l'adresse 0x2f la commande
#(ce qu'on veut lire)
b = bus.read_byte_data(0x2f, 0x58)
# lire un bloc de 16 bytes
b = bus.read_i2c_block_data(0x2f,0x58,16)
bus.write_byte_data(0x2f,offset,value)
# ecrire un bloc de bytes
data = [1,2,3,4,5,6,7,8]
bus.write_i2c_block_data(0x2f,offset,data)
```

SPI

Spécifications

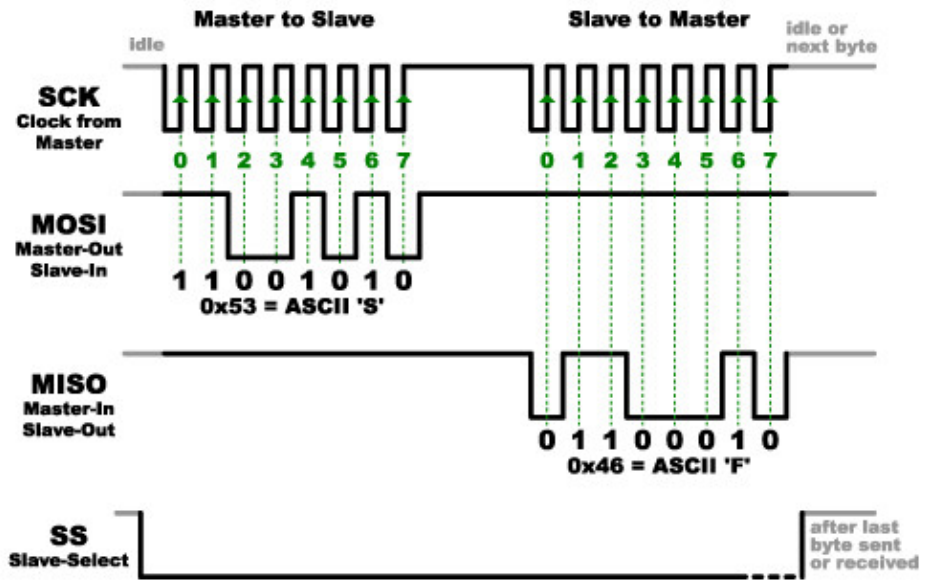
- 3 pins de communication (1 bus), 1 pin par device (sélection de l'esclave)
- SCK : signal d'horloge
- MOSI (Master Out Slave In) : maître $\rightarrow$ esclave
- MISO (Master In Slave Out) : maître $\leftarrow$ esclave
- SS (CS, Slave/Chip Select) : sélection de l'esclave
- Rapide, full-duplex, mais 1 pin par device.



SPI

Spécifications

- 3 pins de communication (1 bus), 1 pin par device (sélection de l'esclave)
- SCK : signal d'horloge
- MOSI (Master Out Slave In) : maître $\rightarrow$ esclave
- MISO (Master In Slave Out) : maître $\leftarrow$ esclave
- SS (CS, Slave/Chip Select) : sélection de l'esclave
- Rapide, full-duplex, mais 1 pin par device.



SPI

En python

```
import spidev
spi = spidev.SpiDev()
spi.open(bus,device)
# definition de la vitesse de transfert
spi.max_speed_hz = 5000
spi.mode = 0b01 #choix de la polarite/phase de SCK
spi.writebytes([0x01,0x02]) #ecrit un tableau byte
val = spi.readbytes(len) # lit len byte
data = [0x01, 0x02, 0x03]
spi.xfer2(data)
spi.close()
```