

# Programmation Hardware

Nicolas Baskiotis

`nicolas.baskiotis@sorbonne-universite.fr`

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

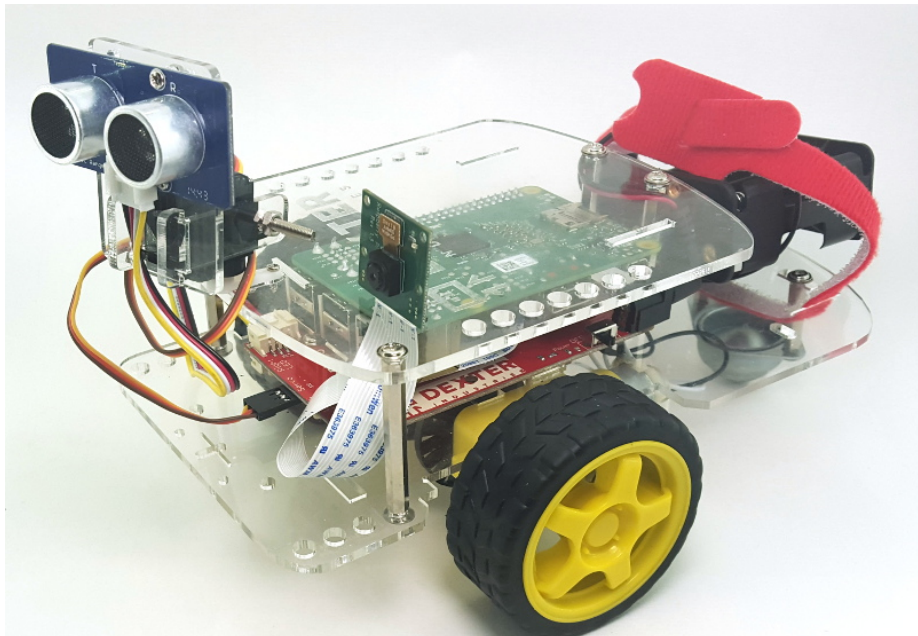
S2 (2023-2024)

# Plan

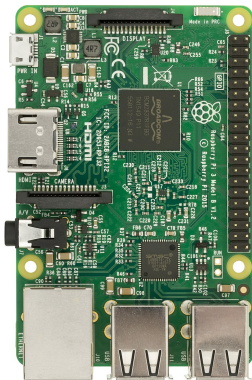
## 1 Description du hardware : GoPiGo Dexter

## 2 Communication hardware

# Vu d'ensemble



# Raspberry Pi



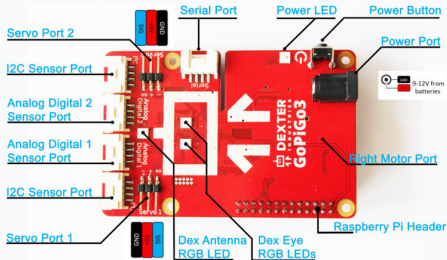
## RPi 3 - Model B Technical Specification

- 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 socket
- camera port for connecting the Raspberry Pi camera
- Micro SD port
- 40pin extended GPIO

# Micro-controller Dexter

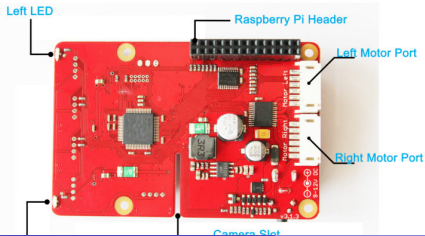
## GoPiGo3

Top Side



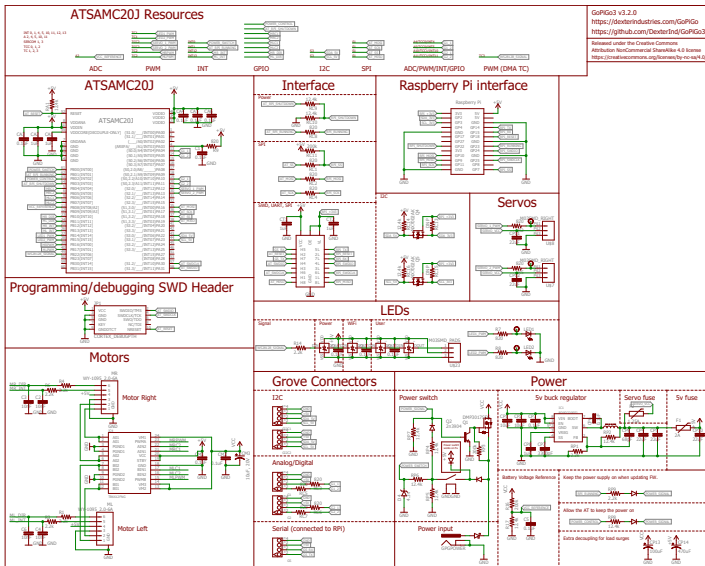
## GoPiGo3

Bottom Side



## 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

1 Description du hardware : GoPiGo Dexter

2 Communication hardware

# 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 !

⇒ 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 numérotation
GPIO.setup(12, GPIO.IN) #configure le pin 12 en lecture
GPIO.setup(13, GPIO.OUT) #configure le pin 13 en écriture
# setup obligatoire, configure pull-up resistance
state = GPIO.input(12) # lecture, 0/1 ou GPIO.HIGH/LOW
GPIO.output(13, GPIO.HIGH) # écriture
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

## Raspberry Pi 3 GPIO Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I <sup>2</sup> C)		DC Power 5v	04
05	GPIO03 (SCL1 , I <sup>2</sup> C)		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 (I <sup>2</sup> C ID EEPROM)		(I <sup>2</sup> C 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

# Du digital à l'analogique

## Problème : le RPi n'a pas de port analogique

- Digital : valeurs discrètes finies (binaire)
  - Analogique : valeurs continues
- ⇒ 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
- ⇒ Comment passer de l'information analogique ?

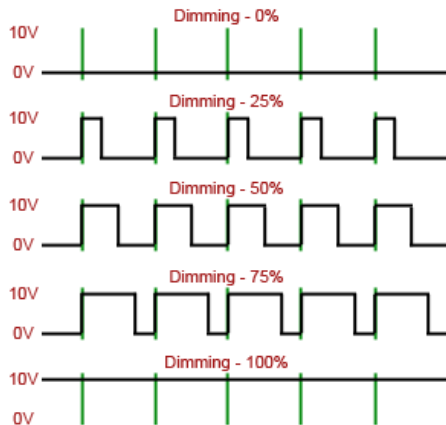
## Encoder l'information

- Convertir les valeurs continues en nombres discrets  
→ compliqué (quand fini un bit ? quand commence un autre ?)
- Convertir en un rapport de 0 à 100%  
→ 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)
```

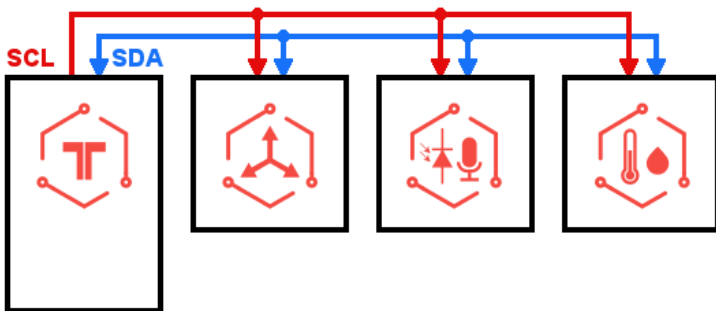
Paquet GPIO :

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

# Bus I2C : Inter-Integrated Circuit

## Spécifications

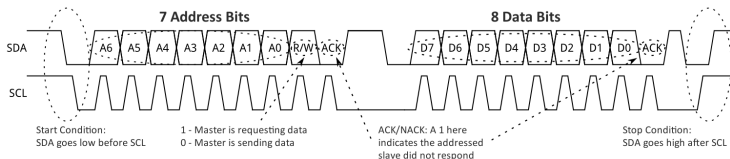
- plusieurs devices sur un simple bus, bi-directionnels
- Protocole Master/Slave : un maître décide qui parle/à qui il parle.
- utilise 2 pins (2 lignes)
  - ▶ Besoin de synchronisation : à quel rythme s'échange 1 bit ?  $\Rightarrow$  Clock signal (signal d'horloge, SCL)
  - ▶ Une ligne de communication,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

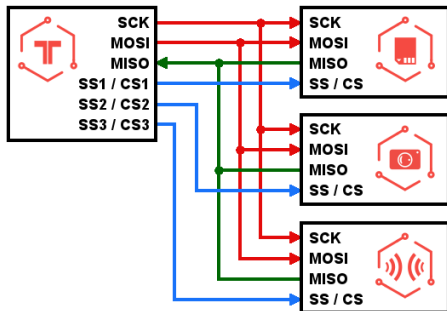
```
from smbus import SMBus
bus = SMBus(0) # numero du bus
#lire du device a l'adresse 0x2f la commande
#(ce qu'on veut lire)
b = bus.read_byte_data(0x2f, 0x58)
# lire un bloc de 16 bytes
b = bus.read_i2c_block_data(0x2f,0x58,16)
bus.write_byte_data(0x2f,offset,value)
# ecrire un bloc de bytes
data =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



# SPI

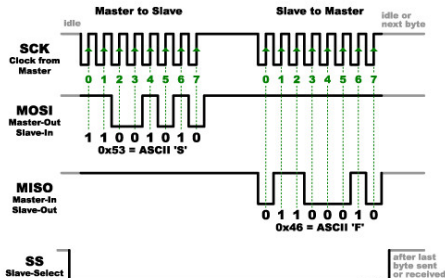
## Spécifications

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



## 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 → esclave
- MISO (Master In Slave Out) : maître ← esclave
- SS (CS, Slave/Chip Select) : sélection de l'esclave
- Rapide, full-duplex, mais 1 pin par device.



## 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()
```