






## Communication I2C avec la Raspberry Pi 3

**Dr. Ing. Chiheb Ameer ABID**

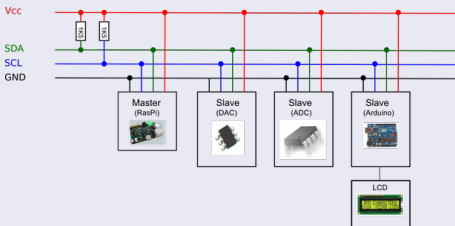
 /in/chiheb-ameur-abid

 chiheb.abid@gmail.com

 Septembre 2024

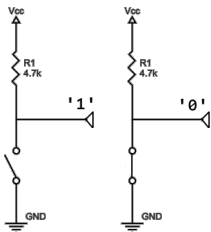
## Le bus I2C en bref

- I2C est un bus permettant la communication synchrone et bidirectionnelle en half-duplex
- Le protocole de communication permet de mettre en communication un composant maître (généralement le microprocesseur) et plusieurs périphériques esclaves
- Plusieurs maîtres peuvent partager le même bus, et un même composant peut passer du statut d'esclave à celui de maître ou inversement.
- La communication n'a lieu qu'entre un seul maître et un seul esclave.
- Chaque esclave possède un identifiant unique (adresse)
  - ☞ Cette adresse est définie par la fabricant au niveau matériel
  - ☞ Certains périphériques permettent de personnaliser une partie de leur adresse



## Aspect techniques du bus I2C

- Le bus I2C repose sur deux lignes pour assurer la communication
  - SDA (Serial Data) est la ligne de données série, utilisée pour transmettre les informations (adresses, données) entre le maître et les esclaves
  - SCL (Serial Clock) est la ligne d'horloge série, générée par le maître pour synchroniser la transmission des données
- Les deux lignes sont de type open-drain en utilisant des résistances PULL-UP
  - Transmission de bit 0 s'effectue en connectant la ligne à la masse
  - Pour transmettre le bit 1, on laisse la ligne à l'état repos : les résistances pull-up assurent que la ligne revient automatiquement à l'état haut



# Communication sur I2C

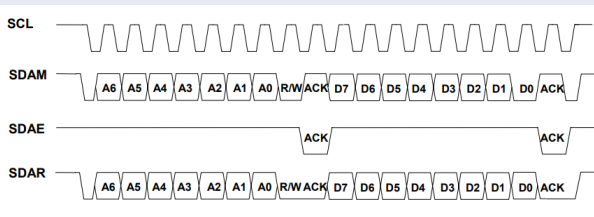
## Principe de communication sur I2C

- Au repos, les deux lignes SDA et SCL sont à '1'
- ① Initialisation du Bus
  - ☞ SCL reste à '1'
  - ☞ SDA passe de '1' à '0'
- ② Sélection de l'esclave
  - ☞ Le maître génère le signal SCL pour synchroniser la transmission
  - ☞ Envoi de l'adresse de l'esclave (7 ou 10 bits)
  - ☞ Envoi d'un bit de Lecture/Écriture
- ③ Acquiescement (ACK) de l'esclave
  - ☞ Le maître libère la ligne SDA pour recevoir l'ACK
  - ☞ L'esclave doit répondre (ACK) en tirant la ligne SDA vers '0'
  - ☞ Si aucun ACK (NACK), la transmission échoue
- ④ Échange de données (Lecture/Écriture)
  - ☞ En mode **lecture**, l'esclave envoie un octet, le maître répond par ACK s'il veut continuer à lire sinon NACK.
  - ☞ Mode **écriture**, le maître envoie un octet, et l'esclave doit répondre par ACK après chaque octet
- ⑤ Arrêt de la communication
  - ☞ Le maître génère la condition STOP en libère la ligne SCL, puis la ligne SDA



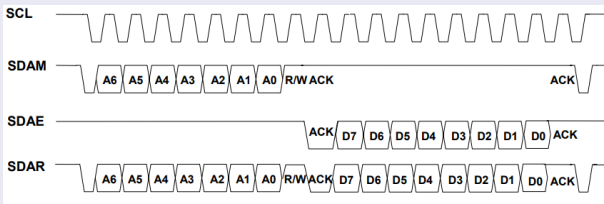
# Communication sur I2C

## Envoie d'une donnée



# Communication sur I2C

## Lecture d'une donnée



# Le bus I2C

## Le bus I2C sur la RPi3


➤ La RPi3 possède deux bus I2C


### ① Le bus I2C0

- 🔌 Broches associées : BCM0(SDA0) et BCM1(SCL0)
- 🔌 Utilisé principalement pour la gestion des HATs (Hardware Attached on Top)
- 🔌 il peut être utilisé pour d'autres périphériques

### ② Le bus I2C1

- 🔌 Broches associées : BCM2(SDA1) et BCM3(SCL1)
- 🔌 Couramment utilisé pour connecter des périphériques externe
- 🔌 Par défaut, I2C1 est désactivé sur Raspberry Pi OS

 ➤ Les broches BCM2 et BCM3 disposent de résistances de pull-up internes (1,8 k $\Omega$ ) vers 3,3 V, ce qui est souvent suffisant pour la plupart des applications I2C

 ➤ Ne pas connecter des périphériques 5 V directement sans un convertisseur de niveau logique vers 3,3 V

# Le bus I2C

## Activation du bus I2C avec Raspberry OS

➤ La RPi3 possède deux bus I2C

① Le bus I2C0

- 🔗 Broches associées : BCM0(SDA0) et BCM1(SCL0)
- 🔗 Utilisé principalement pour la gestion des HATs (Hardware Attached on Top)
- 🔗 il peut être utilisé pour d'autres périphériques

② Le bus I2C1

- 🔗 Broches associées : BCM2(SDA1) et BCM3(SCL1)
- 🔗 Le plus couramment utilisé pour connecter des périphériques externe
- 🔗 Par défaut, I2C1 est désactivé sur Raspberry Pi OS

# Le bus I2C

## Communication I2C avec pigpio

- Initialiser la communication sur le bus I2C pour un périphérique

```
int i2cOpen(unsigned i2cBus, unsigned i2cAddr, unsigned i2cFlags);
```

- `i2cBus` :  $\geq 0$  ; spécifie le bus i2c à utiliser
- `i2cAddr` : 0-0x7F ; spécifie l'adresse du périphérique sur le bus i2c
- `i2cFlags` : doit être à 0 (pas de drapeaux définis)
- Retourne un handle  $\geq 0$  si OK, sinon `PI_BAD_I2C_BUS`, `PI_BAD_I2C_ADDR`, `PI_BAD_FLAGS`, `PI_NO_HANDLE` ou `PI_I2C_OPEN_FAILED`

- Terminer la communication sur le bus i2c avec le périphérique ayant le handle spécifié

```
int i2cClose(unsigned handle);
```

- `handle` : le handle  $\geq 0$  renvoyé par `i2cOpen()`

# Le bus I2C

## Communication I2C avec pigpio

### ↳ Envoyer des données vers un périphérique

```
1 int i2cWriteQuick(unsigned handle, unsigned bit); // Un seul bit
2 int i2cWriteByte(unsigned handle, unsigned bVal); // Un seul octet
3 int i2cWriteDevice(unsigned handle, char *buf, unsigned count); // Envoyer les données
   pointées par buf (jusqu'à 32 octets)
4 int i2cWriteByteData(unsigned handle, unsigned i2cReg, unsigned bVal); // Écrire un seul
   octet dans un registre
5 int i2cWriteWordData(unsigned handle, unsigned i2cReg, unsigned wVal); // Écrire un mot
   (16 bits) dans un registre
6 int i2cWriteI2CBlockData(unsigned handle, unsigned i2cReg, char *buf, unsigned count); //
   Écrire jusqu'à 32 octets dans un registre
```

🔗 **handle** : le handle  $\geq 0$  renvoyé par `i2c0Open()`

🔗 Renvoie 0 si OK, sinon `PI_BAD_HANDLE`, `PI_BAD_PARAM` ou `PI_I2C_WRITE_FAILED`

# Le bus I2C

## Communication I2C avec pigpio

### ↳ Recevoir des données sur le bus I2C

```
1 int i2cReadByte(unsigned handle); // Un seul octet
2 int i2cReadDevice(unsigned handle, char *buf, unsigned count); // Recevoir jusqu'à 32
  octets
3 int i2cReadByteData(unsigned handle, unsigned i2cReg); // Recevoir un seul octet depuis un
  registre
4 int i2cReadWordData(unsigned handle, unsigned i2cReg); // Recevoir un mot (16 bits) depuis
  un registre
5 int i2cReadI2CBlockData(unsigned handle, unsigned i2cReg, char *buf, unsigned count); //
  Recevoir jusqu'à 32 octets depuis un registre
```

- 📌 **handle** : le handle  $\geq 0$  renvoyé par `i2cOpen()`
- 📌 Renvoie  $\geq 0$  si OK, sinon `PI_BAD_HANDLE`, `PI_BAD_PARAM` ou `PI_I2C_READ_FAILED`



## Lecture de température avec MCP9808

## Les registres du capteur MCP9808

- Les registres sont de tailles de 8 et 16 bits
- Registre de configuration (16 bits) : adresse 0x01

U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
—	—	—	—	—	T <sub>HYST</sub>		SHDN
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
Crit. Lock	Win. Lock	Int. Clear	Alert Stat.	Alert Cnt.	Alert Sel.	Alert Pol.	Alert Mod.
bit 7							bit 0

bit 8      **SHDN**: Shutdown Mode bit  
 0 = Continuous conversion (power-up default)  
 1 = Shutdown (Low-Power mode)

- Par défaut, la résolution est à 0,0625°C. Elle peut être modifiée à travers le registre ayant l'adresse 0x08 (8 bits)
- La valeur de la température est retournée dans le registre 0x05 (16 bits) ayant la structure suivante :

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
T <sub>A</sub> vs. T <sub>CRIT</sub> <sup>(1)</sup>	T <sub>A</sub> vs. T <sub>UPPER</sub> <sup>(1)</sup>	T <sub>A</sub> vs. T <sub>LOWER</sub> <sup>(1)</sup>	SIGN	2 <sup>7</sup> °C	2 <sup>6</sup> °C	2 <sup>5</sup> °C	2 <sup>4</sup> °C
bit 15							bit 8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
2 <sup>3</sup> °C	2 <sup>2</sup> °C	2 <sup>1</sup> °C	2 <sup>0</sup> °C	2 <sup>-1</sup> °C	2 <sup>-2</sup> °C <sup>(2)</sup>	2 <sup>-3</sup> °C <sup>(2)</sup>	2 <sup>-4</sup> °C <sup>(2)</sup>
bit 7							bit 0

La valeur de la température est donnée en complément à 2 sur les 13 bits du plus faible poids.

## Lecture de température avec MCP9808

### Identification de l'adresse du capteur

- ↳ Installer l'outil `i2c-tools`

```
sudo apt install i2c-tools
```

- ↳ Lister les bus `i2c` accessibles

```
i2cdetect -l
```

- ↳ Lister les adresses des périphériques connectés au bus `i2c` spécifié

```
i2cdetect -y 1
```

## Programme C++ sur la RPi3

```
1 #include <iostream>
2 #include <pigpio.h>
3 #include <chrono>
4 #include <thread>
5 using namespace std::chrono_literals;
6 int main() {
7     if (gpioInitialise() < 0) {
8         std::cerr << "Error initializing pigpio...\n"; exit(-1);
9     }
10    auto sensorHandle {i2cOpen(1, 0x18, 0)};
11    if (sensorHandle < 0) {
12        std::cerr << "Error opening I2C bus...\n"; exit(-1);
13    }
14    // Configuration : pas d'alerte, mode continu, résolution par défaut (0.0625C)
15    i2cWriteWordData(sensorHandle, 0x01, 0x0000);
16    std::this_thread::sleep_for(1s); // Pause pour laisser le capteur s'initialiser
17    while (true) {
18        auto v {i2cReadWordData(sensorHandle, 0x05)}; // Lecture du registre de température
19        // Conversion des octets : données reçues en Big Endian
20        auto val {((v << 8) | (v >> 8)) & 0x1FFF}; // Masquage pour garder 13 bits
21        // Gestion des températures négatives (bit de signe à la position 12)
22        if (val & (1 << 12)) val -= (1 << 13); // Conversion en valeur signée
23        auto cTemp {val * 0.0625f}; // // Conversion en degrés Celsius
24        std::cout << "Temperature : " << cTemp << "°C" << std::endl;
25        std::this_thread::sleep_for(1s); // Attente avant la prochaine lecture
26    }
27    gpioTerminate();
28    return 0;
29 }
```

**Merci pour votre attention**

