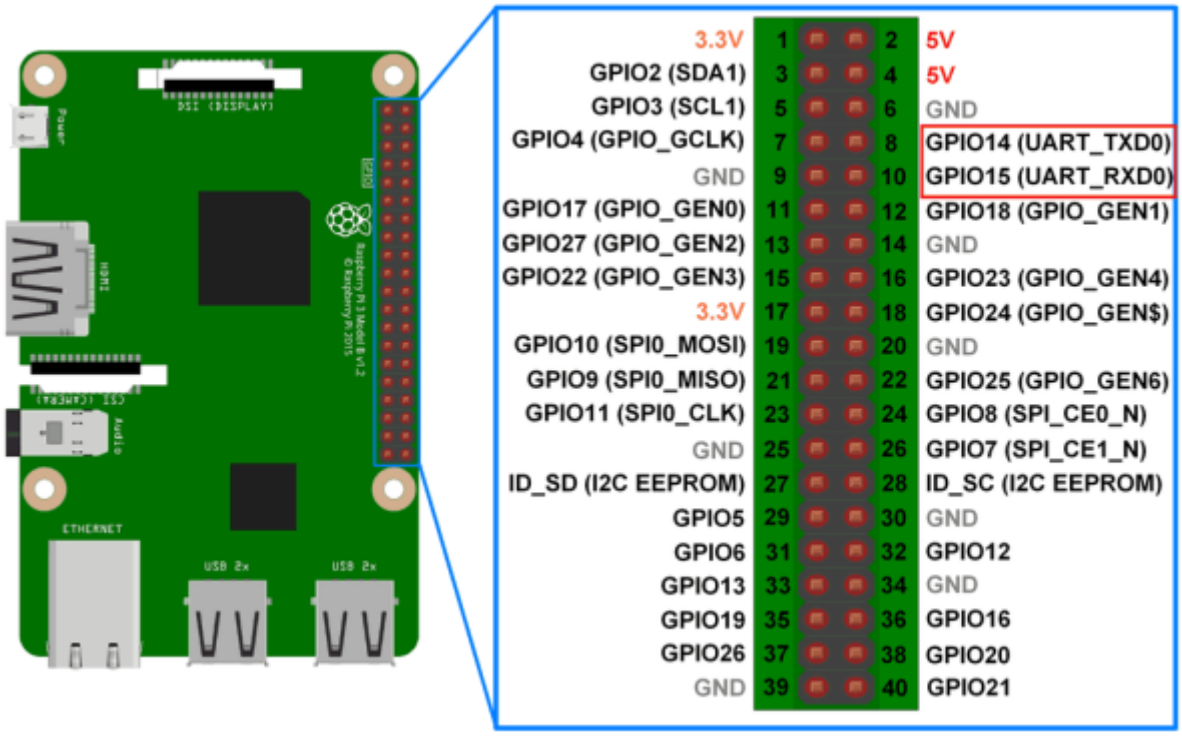
Connexion RPI et ESP32 via UART

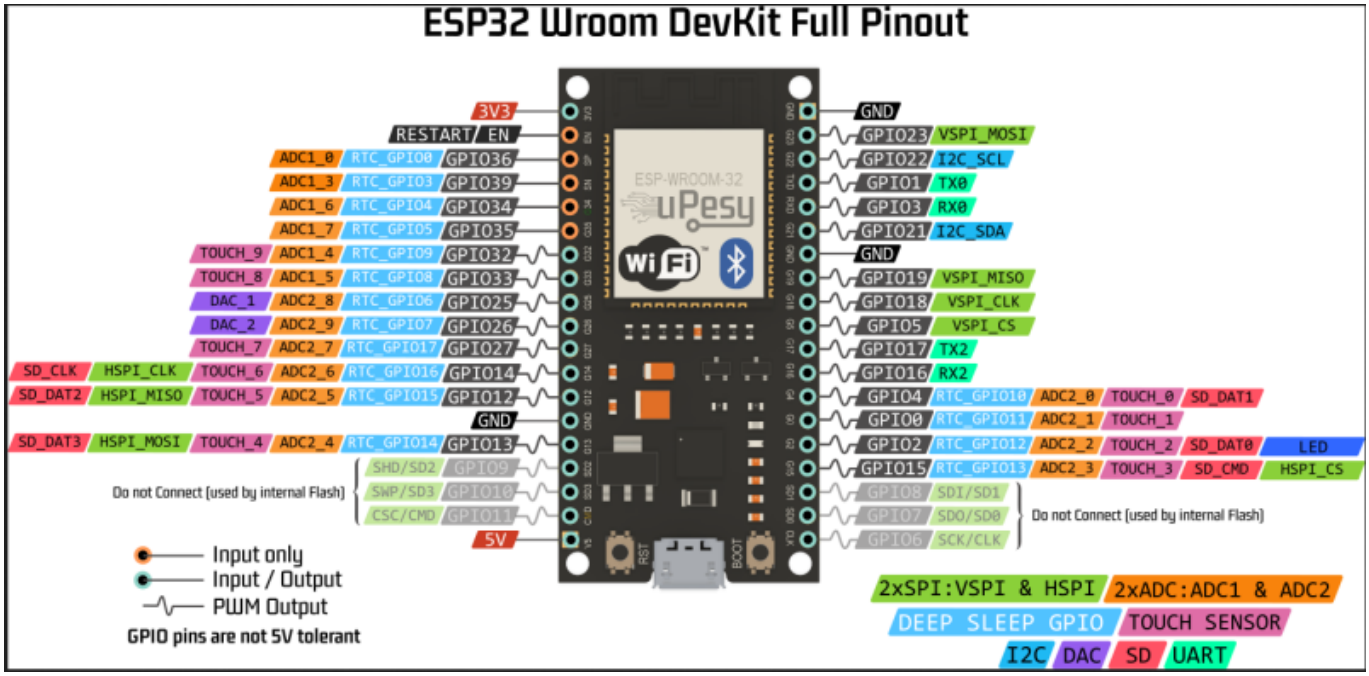
Pour connecter un Raspberry Pi 4 à un ESP32 en utilisant le protocole **UART (série)**, voici les étapes détaillées :

Matériel nécessaire :

- Raspberry Pi 4 - ESP32 - Câbles de connexion (Dupont) - Optionnel : Résistances (si nécessaire pour les niveaux de tension)

Connexion physique :





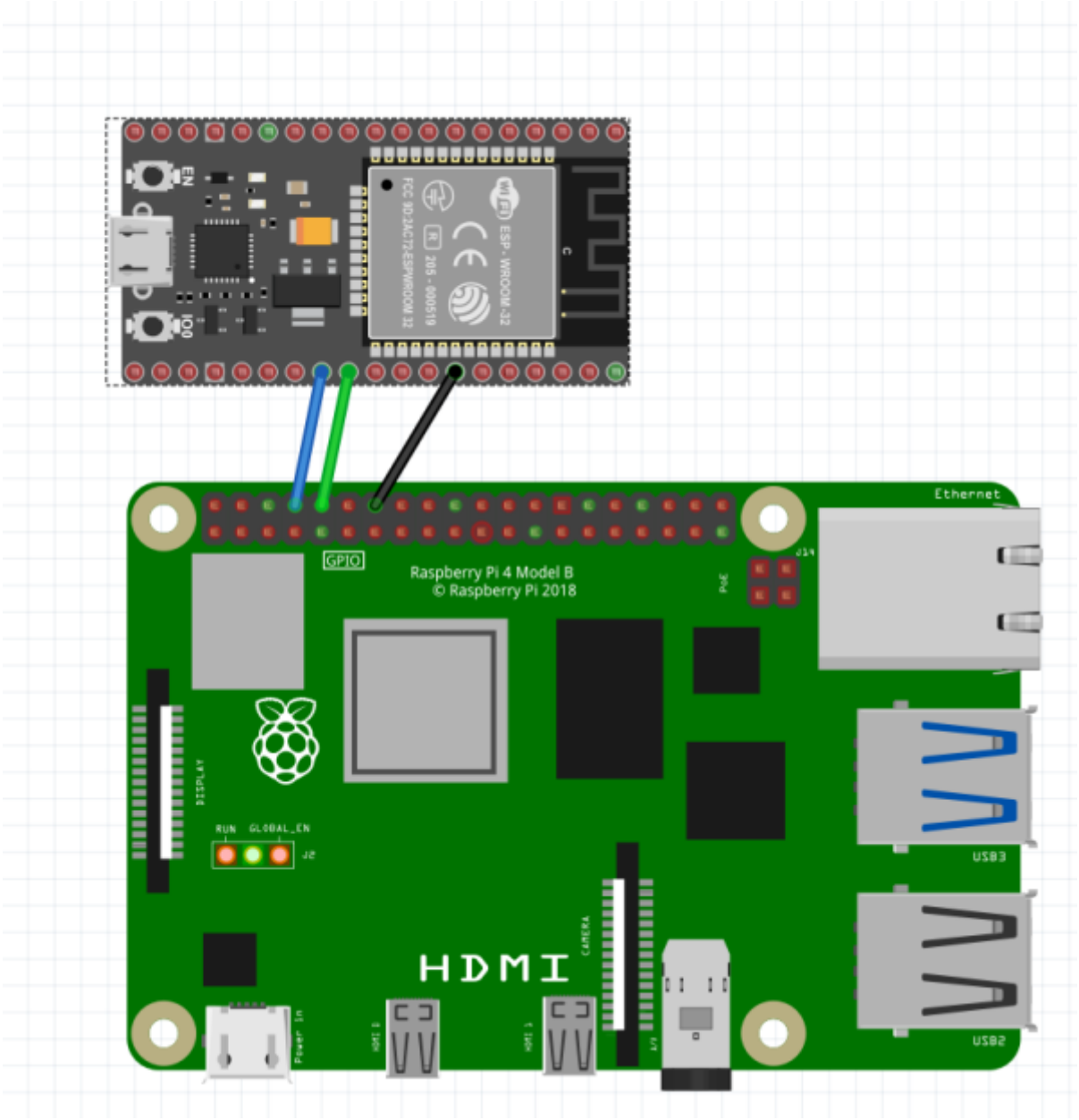
Raspberry Pi 4 → ESP32:

- Raspberry Pi 4 :
  - GPIO 14 (TX) → **ESP32 GPIO 16 (RX)** (Transmission du Raspberry Pi vers l'ESP32) UART(2)
  - GPIO 15 (RX) → **ESP32 GPIO 17 (TX)** (Réception du Raspberry Pi depuis l'ESP32) UART(2)

OU

- GPIO 14 (TX) → **ESP32 GPIO 25 (RX)** (Transmission du Raspberry Pi vers l'ESP32) UART(1)
- GPIO 15 (RX) → **ESP32 GPIO 26 (TX)** (Réception du Raspberry Pi depuis l'ESP32) UART(1)

- Alimentation :
- 3.3V (Raspberry Pi 4) → **3.3V (ESP32)** (L'ESP32 fonctionne en 3.3V, évitez le 5V)
- GND → GND



Important :

1. Vous devez connecter TX à RX et RX à TX.
2. Le Raspberry Pi utilise un niveau logique de 3.3V, donc assurez-vous que l'ESP32 soit alimenté en 3.3V également. Si vous branchez un câble GPIO directement à un autre périphérique qui fonctionne à 5V (par exemple, certaines cartes Arduino), vous risquez d'endommager les broches.

Configuration sur le **Raspberry Pi** (UART Master) :

1. **Activer le port série** :

1. Par défaut, le port série du Raspberry Pi est réservé à la console. Vous devez le libérer pour l'utiliser pour la communication série.

- Ouvrez une terminal et tapez :

exemple003.sh

```
sudo raspi-config
```

1. Allez dans **“Interfacing Options”** puis **“Serial”**. Désactivez l'accès à la console série et activez l'interface série.
2. Redémarrez le Raspberry Pi.
3. Desactiver le Bluetooth sur le Raspberry Pi
4. Installer le module pip
5. Installer PySerial
6. Reboot

2. **Vérifiez que le port série fonctionne** :

1. Une fois le port série activé, vous pouvez vérifier si le périphérique série est détecté. Tapez la commande suivante pour vérifier :

exemple004.sh

```
ls /dev/ttyAMA*
```

Vous devriez voir quelque chose comme `/dev/ttyAMA0`.

3. **Installer les outils de communication série (si nécessaire)** :

Si vous souhaitez envoyer et recevoir des données en ligne de commande, installez **minicom** ou **screen** :

exemple005.sh

```
sudo apt-get install minicom
```

4. **Testez la connexion série avec `minicom`** :

1. Utilisez minicom pour tester la communication série en vous connectant au port `/dev/ttyAMA0` :

exemple006.sh

```
minicom -b 115200 -o -D /dev/ttyAMA0
```

Ctrl + A et ensuite X pour sortir de minicom : Ctrl + A et ensuite Z pour l'Aide : CTRL + A et ensuite O pour configurer minicom

Remarque : Changez le port si nécessaire (en fonction de la sortie de `ls

/dev/ttyAMA0`).

Configuration sur l'**ESP32** (UART Slave) :

1. **Utiliser l'IDE Arduino pour programmer l'ESP32** :

   1. Si vous n'avez pas encore installé le support pour l'ESP32 dans l'IDE Arduino, allez dans `Outils` → `Carte` → Sélectionnez votre modèle ESP32.
   2. Assurez-vous que vous avez installé le paquet ESP32 dans le **Gestionnaire de cartes** de l'IDE Arduino.

2. **Code pour l'ESP32 (réception et envoi UART)** :

-1-Voici un exemple de code pour configurer l'ESP32 pour communiquer via UART(2) :

exemple11.ino

```
#define RXD2 16
#define TXD2 17

#define GPS_BAUDS 115200

HardwareSerial mySerial2(2);

int counter = 0;

void setup() {
  Serial.begin(115200);
  mySerial2.begin(GPS_BAUDS, SERIAL_8N1, RXD2, TXD2);
  Serial.println("Serial 2 demarre en 115200 Bds");
}

void loop() {
  while (mySerial2.available() > 0) {
    char gpsData1 = mySerial2.read();
    Serial.print(gpsData1);
    }
  delay(2000);
  Serial.println("-----------------------------------------");
  mySerial2.println(String(counter));
  Serial.println("Envoie UART2:  " + String(counter));
  counter++;
  delay(2000);
}
```

Ce code permet à l'ESP32 de lire les caractères envoyés par le Raspberry Pi et de répondre avec un

message.

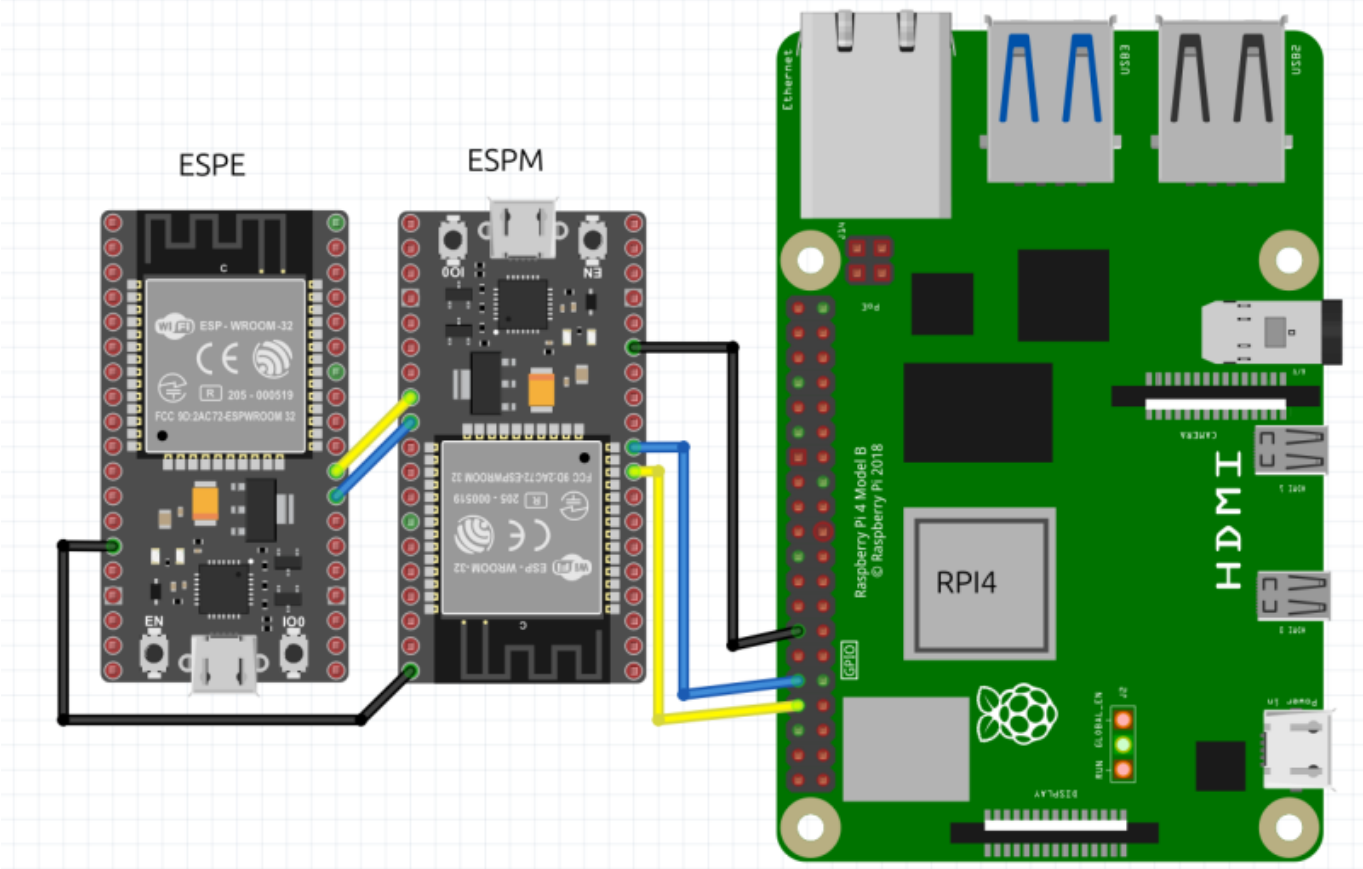
-2- Voici un exemple de code pour configurer l'ESP32 pour communiquer via UART(1) :

exemple12.ino

```
#define RXD1 25
#define TXD1 26

#define GPS_BAUDS 115200

HardwareSerial mySerial1(1);

int counter = 0;

void setup() {
  Serial.begin(115200);
  mySerial1.begin(GPS_BAUDS, SERIAL_8N1, RXD1, TXD1);
  Serial.println("Serial 1 25R 26T demarre en 115200 Bds");
}

void loop() {
  while (mySerial1.available() > 0) {
    char gpsData2 = mySerial1.read();
    Serial.print(gpsData2);
    }
  delay(2000);
  Serial.println("-----------------------------------------");
  mySerial1.println(String(counter));
  Serial.println("Envoie UART1:  " + String(counter));
  counter++;
  delay(2000);
}
```

Ce code permet à l'ESP32 de lire les caractères envoyés par le Raspberry Pi et de répondre avec un message.

Code pour l"ESP32M qui recoit sur l'UART1 le RPI et sur l'UART2 l'ESP32E



Raspberry Pi 4 → ESP32M ←- ESP32E:

- Raspberry Pi 4 :
  - GPIO 14 (TX) → **ESP32M GPIO 25 (RX)** (Transmission du Raspberry Pi vers l'ESP32M) UART(1)
  - GPIO 15 (RX) ←- **ESP32M GPIO 26 (TX)** (Réception du Raspberry Pi depuis l'ESP32M) UART(1)
- ESP32M
  - GPIO 16 (RX) –> **ESP32E GPIO 17 (TX)** (Transmission de l'ESP32M vers l'ESP32E) UART(2)
  - GPIO 17 (TX) ←- **ESP32E GPIO 16 (RX)** (Réception de l'ESP32M depuis l'ESP32E) UART(2)

- Alimentation :
- 3.3V (Raspberry Pi 4) → **3.3V (ESP32M et E)** (L'ESP32 fonctionne en 3.3V, évitez le 5V)
- GND → GND

exemple14.ino

```
#define RXD1 25
#define TXD1 26
#define RXD2 16
#define TXD2 17

#define GPS_BAUDS 115200
HardwareSerial mySerial1(1);
HardwareSerial mySerial2(2);

int counter = 0;
```

```
void setup() {
  Serial.begin(115200);
  mySerial1.begin(GPS_BAUDS, SERIAL_8N1, RXD1, TXD1);
  Serial.println("Serial 1 25R 26T demarre en 115200 Bds");
  mySerial2.begin(GPS_BAUDS, SERIAL_8N1, RXD2, TXD2);
  Serial.println("Serial 2 16R 17T demarre en 115200 Bds");
}

void loop() {
  while (mySerial1.available() > 0) {
    char gpsData1 = mySerial1.read();
    Serial.print(gpsData1);
    }
  while (mySerial2.available() > 0) {
    char gpsData2 = mySerial2.read();
    Serial.print(gpsData2);
    }
  delay(2000);
  Serial.println("-----------------------------------------");
  mySerial1.println(String(counter));
  Serial.println("Envoie UART1:  " + String(counter));
  mySerial2.println(String(counter));
  Serial.println("Envoie UART2:  " + String(counter));
  counter++;
  delay(2000);
}
```

Code pour test sur ESP32E

exemple15.ino

```
    #define RXD2 16
    #define TXD2 17

    #define GPS_BAUDS 115200

    HardwareSerial mySerial2(2);

    int counter = 0;

    void setup() {
      Serial.begin(115200);
      mySerial2.begin(GPS_BAUDS, SERIAL_8N1, RXD2, TXD2);
      Serial.println("Serial 2 demarre en 115200 Bds");
    }
```

```
    void loop() {
      while (mySerial2.available() > 0) {
        char gpsData1 = mySerial2.read();
        Serial.print(gpsData1);
        }
      delay(2000);
      Serial.println("-----------------------------------------");
      mySerial2.println(String(counter));
      Serial.println("Envoie UART2:  " + String(counter));
      counter++;
      delay(2000);
    }
```

Étapes sur le **Raspberry Pi** (UART Master) :

1. **Code Python pour envoyer/recevoir des données via UART** :

Voici un exemple de code Python pour communiquer avec l'ESP32M en utilisant le port série /dev/ttyAMA0 :

Installer le module pyserial sur raspbery :

exemple007.sh

```
sudo apt-get install python3-serial
```

exemple010.py

```
import serial
import time

# Configurer le port série pour le Raspberry Pi
ser = serial.Serial('/dev/ttyAMA0', 115200)  # Le port série, le même que pour Minicom
time.sleep(2)  # Attendre que la communication soit stable

# Envoyer un message à l'ESP32
ser.write(b"Hello ESP32!\n")
print("Envoie Hello ESP32!\n")
# Reception des messages de l'ESP32
while True:
    if ser.in_waiting > 0:  # Si des données sont reçues
        #received = ser.readline().decode('utf-8').strip()  # Lire et décoder les données reçues de l'ESP32 en utf-8
        received = ser.readline().decode('iso-8859-1').strip()  # Lire et décoder les données reçues de l'ESP32 en iso-8859-1
        print("Reçu de l'ESP32:", received)
```

```
    time.sleep(1)
     # Envoie des messages de l'ESP32
    ser.write(b"Hello ESP32!\n")
    print("Envoie Hello ESP32!\n")
```

Lancer le programme python sur le raspberry

test001.sh

```
python3 exemple010.py
```

Ce script Python envoie un message à l'ESP32 et attend la réponse. Vous pouvez tester la communication en lisant les réponses dans le

terminal du Raspberry :

Terminal ESP32:

Vérification et dépannage :

- Si la communication ne fonctionne pas, assurez-vous que les connexions sont correctes, que le code est bien téléchargé sur l'ESP32 et que le Raspberry Pi utilise le bon port série. - Si vous avez des problèmes avec le niveau de tension, vous pouvez utiliser un convertisseur logique pour passer du 3.3V à un 5V, mais ce n'est pas nécessaire si vous utilisez 3.3V des deux côtés. - Si vous ne voyez rien sur le port série (pour le Raspberry Pi), vous pouvez essayer d'utiliser `dmesg | grep tty` pour voir les messages du système concernant les ports série.

Cela devrait vous permettre d'établir une communication UART entre votre Raspberry Pi 4 et votre ESP32.

Test la vitesse entre RPI4 et ESP32 sur UART à 115200Bds

Programme python sur le RPI4

exemple001.py

```
import serial
import time

# Configurer le port série pour le Raspberry Pi
ser = serial.Serial('/dev/ttyAMA0', 115200)  # Le port série, le même que pour >
time.sleep(2)  # Attendre que la communication soit stable

# Envoyer un message à l'ESP32
ser.write(b"Hello ESP32!\n")
print("Envoie Hello ESP32!\n")
while True:
    if ser.in_waiting > 0:  # Si des données sont reçues
        received = ser.readline().decode('utf-8').strip()  # Lire et décoder le>
        print("Reçu de l'ESP32:", received)
        time.sleep(0.008)
        ser.write(b"OK!\n")
        #print("Bien recu  du Raspberry!\n")
```

From:
http://chanterie37.fr/fablab37110/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
http://chanterie37.fr/fablab37110/doku.php?id=start:raspberry:uart:uarttoesp32&rev=1740861975

Last update: **2025/03/01 21:46**