PREMIERE GENERALE

NUMERIQUE ET SCIENCES INFORMATIQUES

CARTE DE DEVELOPPEMENT RASPBERRY PICO W

1 – PRESENTATION DE LA CARTE DE DEVELOPPEMENT RASPBERRY PICO W

1.1 – SoC RP2040

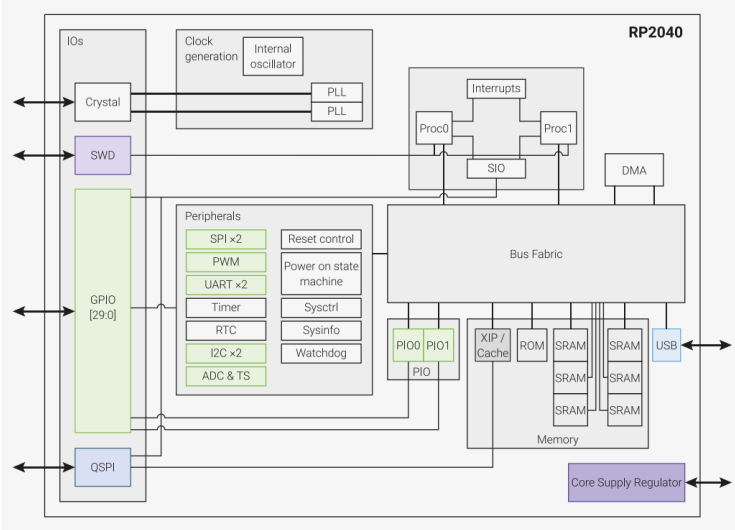
La carte Raspberry PICO W est une carte à microcontrôleur. Cette carte intègre le SoC **RP2040** crée par la fondation Raspberry. Cette carte est dédiée à l'internet des objets (IoT) et plus particulièrement les communications sans fil Wi-Fi pour un coût réduit.

- RP2040 -

Un SoC est un système complet embarqué, sur une seule puce, et constitué de mémoire, d'un ou plusieurs microprocesseurs, de périphériques d'interface, ou tout autre composant nécessaire à la réalisation d'une fonction attendue. Selon les applications, ils peuvent intégrer de la logique, de la mémoire flash, EEPROM, des capteurs, des systèmes opto-électroniques, chimiques ou biologiques ou des circuits de radiocommunication…

Les principaux éléments contenus dans le SoC **RP2040** sont :

- Processeur ARM Cortex M0 à double cœur
- Horloge 133 Mhz.
- 2 Mo de mémoire Flash.
- 264 ko de RAM.
- 2 interfaces UART.
- 2 interfaces SPI.
- 2 contrôleurs I2C.
- 16 canaux PWM.
- 3 ADC 12 bits.
- Un module Wifi.



- Constitution du SoC RP2040 -

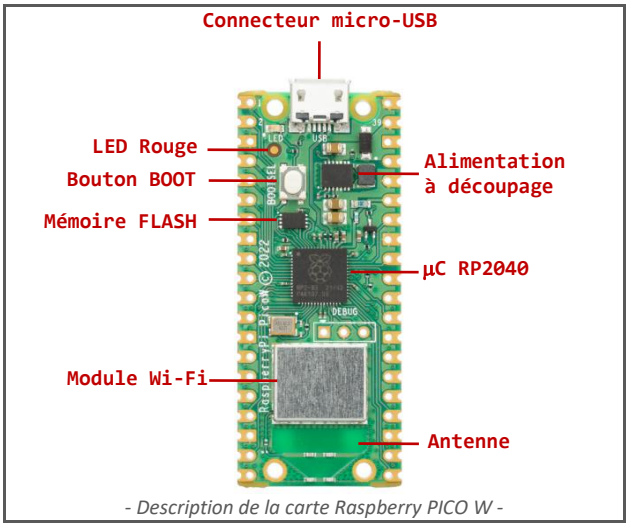
1.2 – Carte de développement Raspberry PICO W

La carte Raspberry PICO W possède des connecteurs latéraux permettant d'accéder aux broches du µC RP2040.

- Carte Raspberry PICO W -

Les principaux composants de la carte **Raspberry PICO W** sont :

- Un **µC RP2040**.
- Un **connecteur micro-USB** qui permet l'alimentation de la carte et la connexion avec un ordinateur.
- Un module et une antenne **Wi-Fi**.
- Une **alimentation à découpage** qui permet de générer la tension d'alimentation de la carte.
- Un **bouton BOOT** qui permet de flasher le micrologiciel.
- Une **DEL** connectée à la broche E/S n°25 de **µC RP2040**.



- Description de la carte Raspberry PICO W -

1.3 – Caractéristiques

Caractéristiques de la carte Raspberry PICO W :

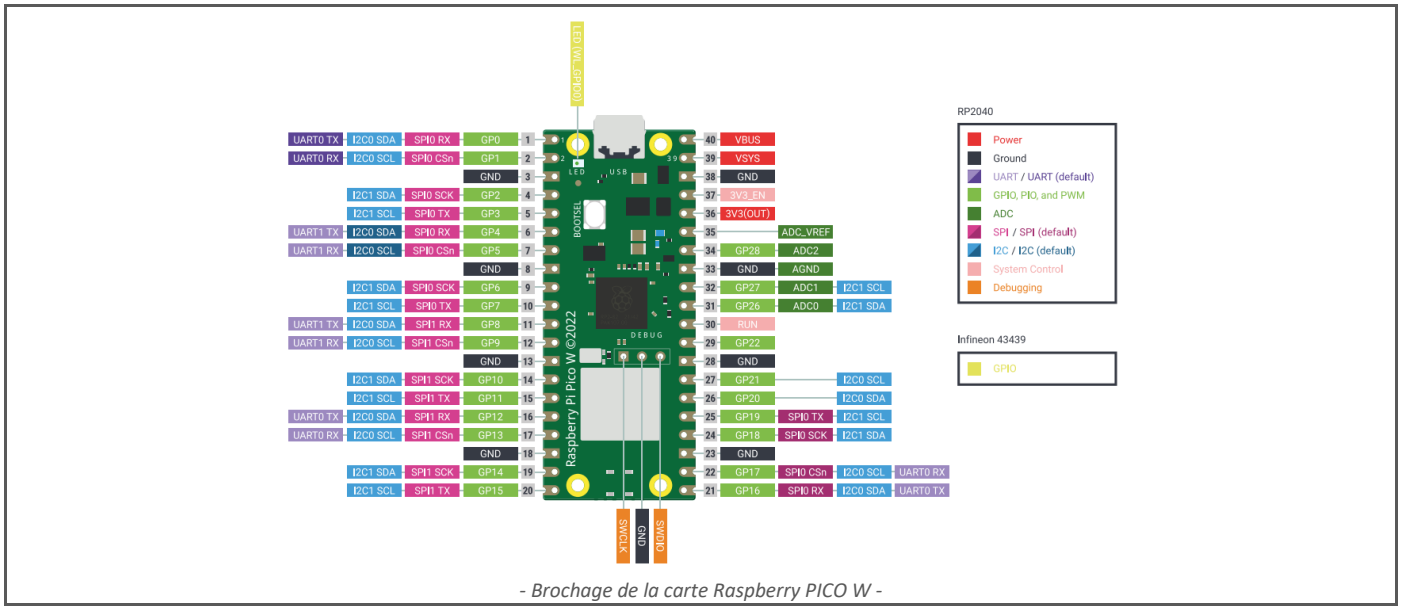
Alimentation	**5 V** par le connecteur micro-USB **1.8 V à 5.5 V** par la broche **VSYS**
Tension de fonctionnement	**3.3 V**
Consommation	**90 à 100 mA** en fonctionnement **1.5 à 2 mA** en veille
Courant maximal délivré par broche	Réglable à **2 mA**, **4 mA** (par défaut), **8 mA**, et **12 mA**
Courant délivré max	**50 mA** sur l'ensemble des broches
Température de fonctionnement	**-20 à 85 °C**
Dimensions	**21 x 51 x 3.91 mm**

Principales caractéristiques du µC RP2040 :

Fréquence	**133 MHz**
Mémoires	SRAM **264 ko** Flash **2 Mo**
E/S disponibles	**23** E/S numériques **3** entrées analogiques (ADC)
Interfaces séries	**2** interfaces I2C **2** interfaces SPI **2** interfaces UART
Interface Wifi	**802.11 b/g/n 2,4 GHz**
Module RTC	**1**
PWM	**16** sorties PWM

1.4 – Brochage de la carte Raspberry PICO W

Le µC PR2040 possède des **ports GPIO** (General Purpose Input/Output) qui sont les broches pouvant être utilisées comme entrées ou comme sorties numériques mais qui permettent également d'accéder aux différentes interfaces de communication et autres éléments. Ces broches sont accessibles, sur la carte Raspberry PICO W à partir des connecteurs latéraux.



- Brochage de la carte Raspberry PICO W -

1.5 – Programmation de la carte Raspberry PICO W

Il existe plusieurs solutions permettant de programmer la carte Raspberry PICO W :

- Avec **MicroPython**, une version du langage Python adaptée à la programmation des µC.
- En **langage C++** avec logiciel Arduino IDE.
- En **langage C/C++** avec un SK dédié.

2 – PROGRAMMATION DE LA CARTE RASPBERRY PICO EN MICROPYTHON

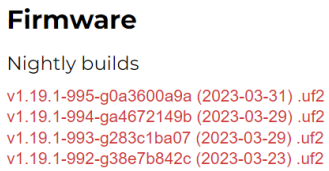
2.1 – Présentation de MicroPython

MicroPython est un interpréteur Python optimisé pour les microcontrôleurs comme les cartes Raspberry PICO. Les scripts Python sont directement exécutés sur les cartes Raspberry PICO. Pour cela, il suffit de flasher la carte Raspberry PICO avec MicroPython et d'utiliser un IDE Python (par exemple Thonny IDE) pour éditer des scripts Python et les transférer sur la Raspberry PICO.

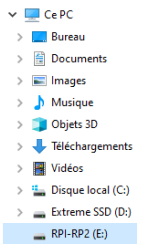
2.2 –Installer MicroPython sur une carte Raspberry PICO avec l'IDE Thonny

Pour flasher MicroPython sur une carte Raspberry PICO avec l'IDE Thonny il faut suivre les étapes suivantes :

1 Télécharger la dernière version stable du firmware MicroPython (fichier **`.uf2`**) sur le site : https://micropython.org/download/rp2-pico-w/



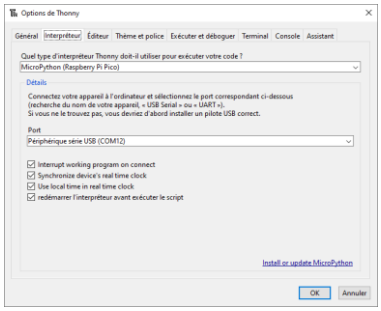
2 Maintenir le bouton **`BOOSEL`** appuyé et connecter la carte Raspberry PICO W à l'ordinateur. La carte Raspberry PICO W apparaît alors dans l'explorateur de fichiers comme une clé USB classique qui s'appelle **RPI-RP2**. Copier/coller le firmware MicroPython (fichier **`.uf2`**) téléchargé dans le dossier **`RPI-RP2`**. Le dossier se ferme et la carte Raspberry PICO W redémarre sur MicroPython.



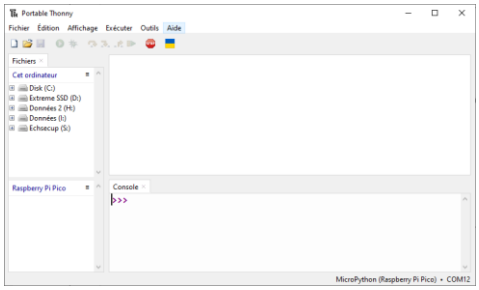
3 Lancer l'IDE Thonny.

4 Cliquer sur « **Sélectionner l'interpréteur…** » dans le menu « **Exécuter** ».

5 Choisir l'interpréteur « **MicroPython (Raspberry Pi Pico)** » et indiquer le port COM utilisé par la carte Raspberry PICO puis cliquer sur « **OK** ».



6 La carte Raspberry PICO apparait alors dans l'onglet « **Fichiers** » de Thonny.



3.3 – Premier programme python sur la carte Raspberry PICO

Maintenant que tous les outils sont installés, on va pouvoir créer un premier programme et l'envoyer sur la carte Raspberry PICO.

Editer le programme suivant :

```
from machine import Pin
import time

pin_led = Pin('LED', mode = Pin.OUT) # La broche de la LED built-in est placée en sortie

while True:
    pin_led.value(True)
    time.sleep(1)
    pin_led.value(False)
    time.sleep(1)
```

Deux possibilités s'offrent à nous :

1. Le script est enregistré dans l'ordinateur :

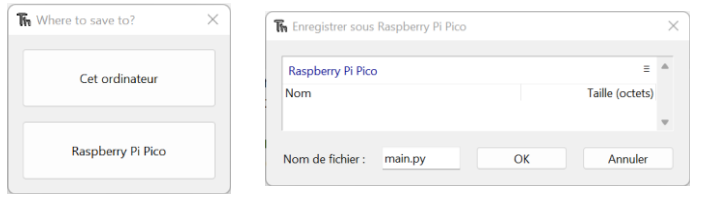
Enregistrer le dans « **cet ordinateur** » sous le nom que l'on veut.

- Enregistrement du fichier dans l'ordinateur -

2. Le script est enregistré dans la carte Raspberry.

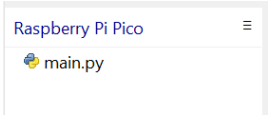
Enregistrer le dans l'« **appareil MicroPython** » sous le nom « **main.py** ».

Le fait de nommer le fichier « **main.py** », permettra de lancer automatiquement le programme lors de la mise sous tension de la carte Raspberry PICO.



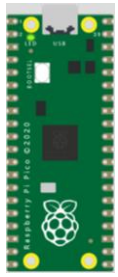
- Enregistrement du fichier dans la carte Raspberry Pico -

Il devrait alors apparaitre dans la liste des fichiers contenus dans la carte Raspberry PICO.



- Contenu de la carte PI PICO -

Pour exécuter le programme il faut cliquer sur « **Exécuter le script courant** » dans le menu « **Exécuter** » ou bien cliquer sur l'icône .



- Résultat exécution -

3 – GESTION DES ENTREES/SORTIES NUMERIQUES DE LA CARTE RASPBERRY PICO

3.1 – Entrées/sorties numériques de la carte Raspberry PICO

Le µC RP2040 dispose de **30 E/S GPIO** mais seules 26 broches GPIO sont disponibles sur la carte Raspberry PICO. Les 4 autres sont réservées à des usages particuliers. Ces 26 broches GPIO pouvant être utilisées comme **E/S numériques**.

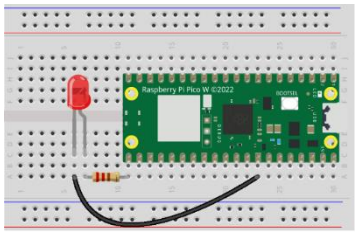
Chaque broche GPIO ne peut délivrer que **4 mA** au maximum, par défaut. Cette valeur est réglable au niveau des registres du microcontrôleur, avec comme choix possibles : **2 mA**, **4 mA**, **8 mA**, et **12 mA**. La somme des courants délivrés par l'ensembles des broches GPIO utilisées ne doit, cependant, pas dépasser **50 mA**.

3.2 – Exemple : Commander une sortie numérique de la carte Raspberry PICO

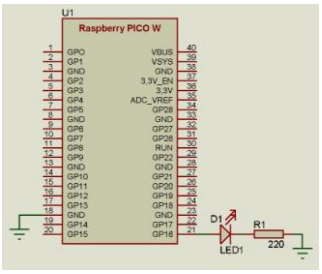
Pour commander une sortie numérique il faut, au préalable configurer la broche numérique en mode « sortie » par l'intermédiaire de l'instruction `nom_broche = Pin(num_pin, PIN.OUT)` en indiquant le numéro de la broche numérique. L'instruction `nom_broche.value(etat)` permettra de placer cette sortie dans l'état logique désiré.

Le programme suivant permet de faire clignoter une LED connectée sur la sortie **GPIO16** :

```
from machine import Pin
import time

pin_led = Pin(16, mode=Pin.OUT)

while True:
    pin_led.value(True)
    time.sleep(1)
    pin_led.value(False)
    time.sleep(1)
```



- Schéma de cablâge -



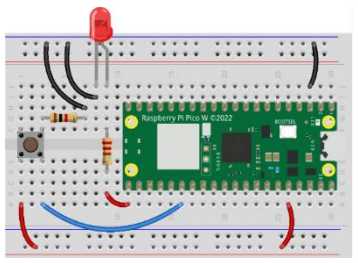
- Schéma struturel -

3.3 – Exemple : Lire l'état d'une entrée numérique de la carte Raspberry PICO

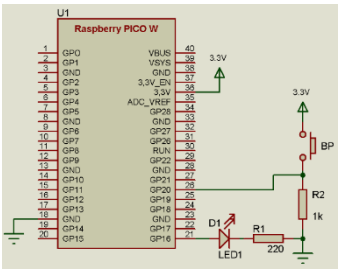
Pour lire l'état logique d'une broche numérique, il faut au préalable configurer la broche numérique en mode « entrée » par l'intermédiaire de l'instruction `nom_broche = Pin(num_pin, PIN.IN)` en indiquant le numéro de la broche numérique. L'instruction `etat = nom_broche.value()` permettra ensuite de récupérer l'état logique de cette entrée.

Le programme suivant permet d'allumer la LED connectée sur la broche **GPIO16** si le bouton poussoir connecté sur la broche **GPIO20** est appuyé.

```
from machine import Pin
import time

pin_bp = Pin(20, mode=Pin.IN)
pin_led = Pin(16, mode=Pin.OUT)

while True:
    if (pin_bp.value() == True):
        pin_led.value(True)          # On allume la led connectée sur GPIO16
    else:
        pin_led.value(False)         # On éteint la led connectée sur GPIO16
```



- Schéma de cablâge -



- Schéma struturel -

4 – GESTION DES ENTREES ANALOGIQUES DE LA CARTE RASPBERRY PICO

4.1 – Entrées analogique de la carte Raspberry PICO

Le SoC RP2040 possède 5 voies analogiques associées à un ADC **16 bits**. Seules 3 voies sont disponibles, les deux autres étant réservées. Sur la carte Raspberry PICO, ces trois voies sont accessibles sur les broches **GPIO26** , **GPIO27** et **GPIO28**.

N° Voie	Disponibilité	N° broche	Rôle
ADC0	Libre	GPIO26	Utilisation externe
ADC1	Libre	GPIO27	Utilisation externe
ADC2	Libre	GPIO28	Utilisation externe
ADC3	Réservé		Mesure de la tension $\mathbf{V_{sys}/3}$
ADC4	Réservé		Mesure de la température interne de µC

Pour utiliser les entrées analogiques, il faut utiliser le sous-module **ADC** de la librairie **machine**.

```
from machine import ADC
```

Il faut ensuite créer en objet ADC en spécifiant le numéro de la broche analogique ou le numéro de la voie.

```
from machine import Pin, ADC
adc_pin = Pin(26, mode=Pin.IN)
adc = ADC(adc_pin)  # Utilisation de la broche GPIO26 (ADC0)
```

```
from machine import ADC
adc = ADC(0)   # Utilisation de la voie ADC0 (GPIO26)
```

Chaque ADC renvoie un nombre numérique **N** de **16 bits** proportionnel à la tension appliquée **Ve** sur l'entrée du convertisseur.

$$N = Ve \times \frac{65535}{3.3}$$

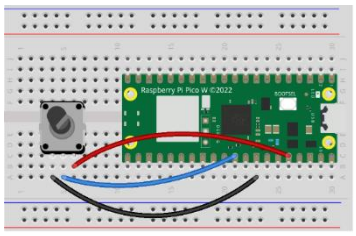
Pour récupérer ce nombre numérique N, il faut utiliser l'instruction suivante :

```
N = adc.read_u16();
```

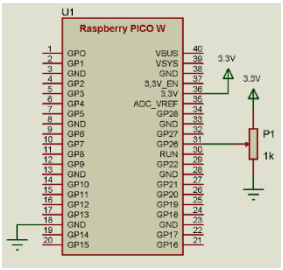
4.2 – Exemple : Mesurer une tension analogique

Le programme suivant permet d'afficher la valeur de la tension aux bornes d'un potentiomètre et appliquée sur la voie **ADC0**.

```
from machine import ADC
import time

adc = ADC(0)   # Utilisation de la voie ADC0 (GPIO26)

while True:
    N = adc.read_u16()
    Ve = N * 3.3 / 65535
    print("N = ", N)
    print("Ve = ", Ve)
    time.sleep(2)
```



- Schéma de cablâge -



- Schéma structurel -

Après avoir exécuté le programme, la valeur de la tension aux bornes du potentiomètre et appliquée l'entrée **ADC0** (**GPIO26**) est affichée toutes les 2 secondes sur la console.

```
Console
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
N =  65535
Ve =  3.3
N =  65535
Ve =  3.3
N =  60094
Ve =  3.02602
N =  47675
Ve =  2.400664
N =  35368
Ve =  1.780948
N =  28743
Ve =  1.447347
```

- Résultat de l'exécution du programme -

4.3 – Exemple : Mesurer une température avec le capteur de température Grove

Le capteur de température Grove délivre une tension analogique fonction de la température mesurée. Ce capteur utilise une thermistance pour mesurer la température ambiante. La résistance de cette thermistance augmente lorsque la température diminue. La plage de températures mesurables par ce capteur est de **-40 à 125°C** et la précision est de **±1,5°C**.

- Module TMP36 -

La valeur de la thermistance **RT** est déterminée à partir de la valeur **N** (entier compris entre 0 et 65535) obtenue lors de la lecture de la tension en sortie du capteur.

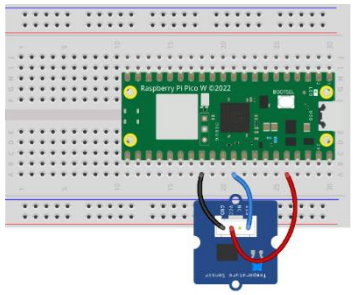
$$RT = R0\left(\frac{65535}{N} - 1\right) \qquad R0 = 100\,k\Omega$$

La valeur de la température est alors obtenue à partir de la relation ci-contre :

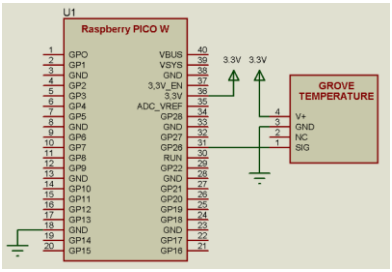
$$T = \frac{1}{\frac{\log\left(\frac{RT}{R0}\right)}{B} + \frac{1}{298.15}} - 273.15 \qquad B = 4275$$

Le programme suivant affiche la valeur de la température mesurée par le capteur connectée sur **ADC0** (**GPIO26**) :

```
from machine import ADC
import time
import math

adc = ADC(0)   # Utilisation de la voie ADC0 (GPIO26)
B = 4275
R0 = 100000

while True:
    N = adc.read_u16()
    Ve = N * 3.3 / 65535
    R = R0 * (65535 / N - 1)
    temp = 1 / (math.log10(R / R0) / B + 1 / 298.15) - 273.15
    print("T = ", temp, " oC")
    time.sleep(2)
```

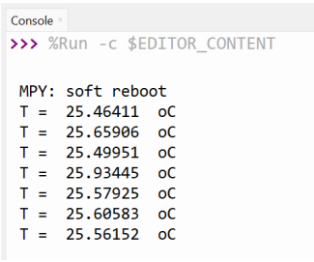


- Schéma de cablâge -



- Schéma struturel -

Après avoir exécuté le programme, la température mesurée par le capteur Grove est affichée toutes les 2 secondes sur la console.



- Résultat d'exécution du programme -

5 – GESTION DES PWM SUR LA CARTE RASPBERRY PICO

5.1 – PWM sur la carte Raspberry PICO

Un signal modulé en largeur d'impulsion (MLI, ou PWM pour Pulse Width Modulation) permet de contrôler l'intensité lumineuse d'une LED, la vitesse de rotation d'un moteur ou encore la position angulaire d'un servomoteur.

Dans la carte Raspberry PICO, les signaux PWM sont générés par des blocs matériels spécifiques. Une fois configurés dans le script, ces blocs génèrent les signaux PWM en permanence. Les ressources du processeur sont alors disponibles pour exécuter d'autres tâches.

Le Raspberry Pi Pico possède 8 paires de blocs PWM (`PWM 0` à `PWM 7`), soit **16 sorties PWM** au total. Ces sorties PWM peuvent, théoriquement, être rattachées à n'importe quelle broche de la carte.
Chaque bloc PWM peut avoir une fréquence indépendante. Sur la base d'un µC RP2040 fonctionnant à `125 MHz`, les fréquences PWM possibles sur les sorties d'un Raspberry Pico sont comprises entre `7.5 Hz` à plusieurs Mégahertz.

L'utilisation de PWM avec MicroPython est très simple. MicroPython fait abstraction du hardware et sélectionne automatiquement un bloc PWM disponible. La configuration consiste à associer un objet `PWM` à une broche et de choisir la fréquence PWM.

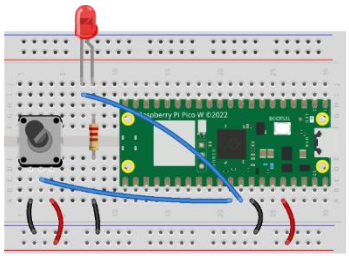
```
from machine import PWM, Pin
nom_pwm = PWM(Pin(numero_broche, mode=Pin.OUT))
nom_pwm.freq(frequence)
```

Pour générer le signal PWM, on dispose de la méthode **`duty_u16`**`(rapport_cyclique)`. Le paramètre `rapport_cyclique` doit être égal à **`0`** pour un rapport cyclique de **`0%`** et **`65535`** pour un rapport cyclique de **`100%`**.

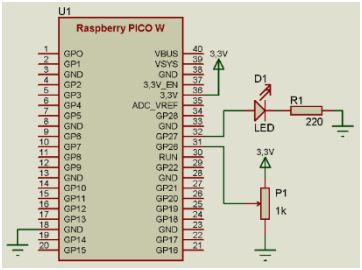
5.2 – Exemple : Commander l'éclairement d'une LED par PWM

Le programme ci-dessous permet de régler l'éclairement de la LED en fonction du réglage du potentiomètre :

```
from machine import ADC, PWM, Pin
import time

adc = ADC(0)
led_pwm = PWM(Pin(27, mode=Pin.OUT))
led_pwm.freq(50)

while True:
    N = adc.read_u16()
    led_pwm.duty_u16(N)
    time.sleep_us(20)
```



- Schéma de cablâge -



- Schéma struturel -

5.3 – Exemple : Commander la position d'un servomoteur

Pour commander un servomoteur, il faut lui fournir des impulsions avec une période de `20 ms`. La position du servomoteur sera fonction de la durée de l'impulsion :

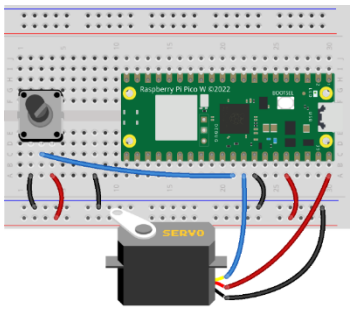
- durée d'impulsion égale à `0,5 ms` (rapport cylique = `2.5%`), le servomoteur se met en position `0°` ;
- durée d'impulsion égale à `1.5 ms` (rapport cylique = `7.5%`), le servomoteur se met en positon `90°` ;
- durée d'impulsion égale à `2,4 ms` (rapport cylique = `12%`), le servomoteur se met en position `180°`.

Le servomoteur utilisé est un servomoteur HITEC HS-422 qui nécessite une alimentation de **4,8** à **6V**. Il est donc nécessaire d'utiliser une alimentation externe pour pouvoir l'alimenter avec une tension suffisante.

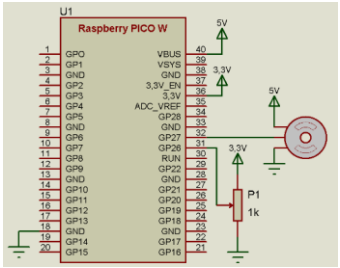
- Servomoteur HS-422 -

Le programme ci-dessous permet de positionner un servomoteur connecté sur la broche `GPIO27` en fonction de la tension générée par un potentiomètre connecté sur la broche `GPIO26`.

```
from machine import ADC, PWM, Pin
import time

adc = ADC(0)
servo_pwm = PWM(Pin(27, mode=Pin.OUT))
servo_pwm.freq(50)
rcy_min = 1640          # rcy = 2,5%
rcy_max = 7864          # rcy = 12%

while True:
    N = adc.read_u16()                  # Lecture de l'entrée GPIO26
    angle = N * 180 // 65535            # angle = f(réglage potentiomètre)
    rcy = int((rcy_max - rcy_min) * angle / 180 + rcy_min)    # Calcul Rapport Cyclique
    pwm_servo.duty_u16(rcy)                                   # Génération signal PWM
    time.sleep_ms(10)
```



- Schéma de cablâge -



- Schéma struturel -

6 – COMMUNICATION EN I2C AVEC LA CARTE RASPBERRY PICO

6.1 – Interfaces I2C de la carte Raspberry PICO

La carte Raspberry PICO dispose de 2 interfaces I2C, nommés `I2C0` et `I2C1` indépendantes l'une de l'autre. Plusieurs broches GPIO permettent d'accéder aux lignes SDA et SCL de ces 2 interfaces.

Interface	Ligne SDA	Ligne SCL
I2C0	GPIO0, GPIO4, GPIO8, GPIO12, GPIO16, GPIO20	GPIO1, GPIO5, GPIO9, GPIO13, GPIO17, GPIO21
I2C1	GPIO2, GPIO6, GPIO10, GPIO14, GPIO18, GPIO26	GPIO3, GPIO7, GPIO11, GPIO15, GPIO19, GPIO27

Les interfaces I2C sont gérées par la classe **I2C** de la librairie **machine**. Il faut créer un objet de type **I2C** sur lequel seront appliqués les différentes méthodes en spécifiant le numéro de l'interface, le numéro des broches **scl** et **sda** et éventuellement la fréquence des signaux I2C.

```
from machine import I2C
nom_I2C = I2C(numero_Interface, scl=Pin(num_broche), sda=Pin(num_broche), freq = frequence)
```

Les principales méthodes de cette librairie sont :

Méthode « start »

Cette méthode génère une condition de START sur le bus. Le signal SDA passe au niveau bas pendant que SCL est au niveau haut.

```
nom_I2C.start()
```

Méthode « stop »

Cette méthode génère une condition de STOP sur le bus. Le signal SDA passe au niveau haut pendant que SCL est au niveau haut.

```
nom_I2C.stop()
```

Méthode « scan »

Scanne toutes les adresses des composants connectés sur l'interface I2C et retourne une liste Python contenant ces adresses.

```
nom_I2C.scan()
```

Méthode « writeto »

Cette méthode envoi les octets disponibles dans `data` vers l'esclave dont l'adresse `adr` est mentionnée en paramètre. Si le paramètre `stop` est placé à `True`, la condition d'arrêt (STOP) est générée à la fin du transfert.

```
nom_I2C.writeto(adr, data, stop = True)
```

Méthode « readfrom »

Cette méthode permet la lecture des `nboctets` octets transmis par l'esclave identifié par son adresses `addr`. Si le paramètre `stop` est à `True` alors la condition de STOP est générée à la fin du transfert

```
nom_I2C.readfrom(adr, nboctet, stop = True)
```

Certains périphériques I2C fonctionnent comme des mémoires (ou ensemble de registres) qu'il est possible de lire ou dans lesquels il est possible d'écrire. Dans ce cas, il y a deux adresses associées avec le périphérique I2C : une adresse I2C et une adresse mémoire. Un certain nombre de méthodes sont conçues pour communiquer avec ce type de périphériques. Parmi lesquelles on trouve :

Méthode « readfrom_mem »

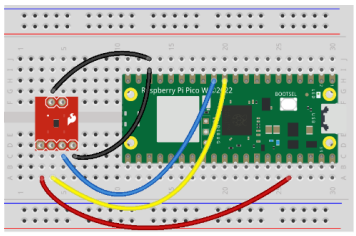
Cette méthode permet la lecture des **nboctets** octets dans la mémoire **memadr** du périphérique identifié par son adresse **addr**.

```
nom_I2C.readfrom_mem(adr, memadr, nboctet)
```

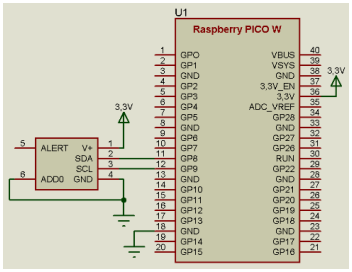
6.1 – Exemple : Affichage de la température mesurée par le capteur TMP102

Le programme ci-dessous permet de lire la température mesurée par le capteur de température tmp102 et de l'afficher dans la console.

```
from machine import I2C, Pin
import time

tmp102_I2C = I2C(0, scl = Pin(9), sda = Pin(8), freq = 400000) # Création d'un objet I2C
tmp102_adresse = 0x48;
ptr_registre = 0;

while True:
    data = tmp102_I2C.readfrom_mem(tmp102_adresse, ptr_registre, 2) # Lecture des données
    octets_tmp = list(data)          # Conversion des données en tableau
    temp = ((octets_tmp[0] * 256 + octets_tmp[1]) >> 4) * 0.0625;    # Calcul température
    print("Température : ", temp, " °C")
    time.sleep(2)
```



- Schéma de cablâge -



- Schéma structurel -

Après avoir exécuter le programme, la température mesurée par le capteur TMP102 est affichée toutes les 2 secondes dans la console.

```
Console
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Température :  27.75  °C
Température :  27.6875  °C
Température :  27.625  °C
Température :  27.5  °C
Température :  27.4375  °C
```

- Résultat d'exécution du programme -

7 – COMMUNICATION EN SPI AVEC LA CARTE RASPBERRY PICO

7.1 – Interfaces SPI de la carte Raspberry PICO

La carte Raspberry PICO intègre 2 interfaces SPI : **SPI0**, **SPI1** indépendantes l'une de l'autre. Plusieurs broches GPIO permettent d'accéder aux lignes des interfaces SPI.

Interface	MOSI	MISO	SCK	CS
SPI0	GPIO3, GPIO7, GPIO19	GPIO0, GPIO4, GPIO16	GPIO2, GPIO6, GPIO18	GPIO1, GPIO5, GPIO17
SPI1	GPIO11, GPIO15	GPIO8, GPIO12	GPIO10, GPIO14	GPIO9, GPIO13

Les interfaces SPI sont gérées par la classe **SPI** de la librairie **machine**. Il faut créer un objet de type **SPI**, sur lequel seront appliqués les différentes méthodes, en spécifiant le numéro de l'interface, la vitesse de communication (**baudrate**) et éventuellement l'état de repos des lignes du bus SPI (**polarity**), le front sur lequel les signaux changent d'état (**phase**). Il faut également définir les numéros des broches utilisées par l'interface.

```
from machine import SPI
nom_SPI = SPI(num_interface, bauderate = vitesse, polarity = etat_repos, phase = etat_front,
sck = Pin(num_pin), mosi = Pin(num_pin), miso = Pin(num_pin))
```

Méthode « read »

Cette méthode lit le nombre d'octets **nbytes** sur la ligne MISO, tout en écrivant l'octet spécifié par l'argument **write** sur la ligne MOSI.

```
nom_SPI.read(nbytes, write = 0x00);
```

Méthode « Write »

Cette méthode écrit les octets stockés dans un objet **data** sur la ligne MOSI. Il ne renvoie rien.

```
nom_SPI.write(data);
```

7.2 – Exemple : Affichage de la pression atmosphérique mesurée par le capteur MPL115A1

La mesure de la pression atmosphérique est réalisée au moyen du composant **MPL115A1**. Il s'agit d'un **baromètre** possédant une **interface SPI**. Sparkfun propose une petite carte équipée d'un **MPL115A1**.

- Capteur MPL115A1 -

La pression atmosphérique sera déterminée à partir de la pression compensée `Pcomp` :

$$\textbf{Pression (kPa)} = \textbf{Pcomp} \times \frac{\textbf{115} - \textbf{50}}{\textbf{1023}} + \textbf{50}$$

La pression compensée P_{comp} peut être calculée à partir des données transmises par le capteur MPL115A1 :

$$\textbf{Pcomp} = \textbf{a0} + (\textbf{b1} + \textbf{c12} \times \textbf{Tadc}) \times \textbf{Padc} + \textbf{b2} \times \textbf{Tadc}$$

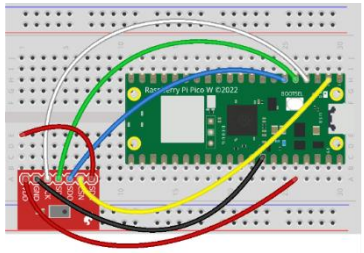
La valeur des coefficients `a0`, `b1`, `b2`, `c11`, `c12` sont spécifiques à chaque capteur. Ces valeurs restent constantes et sont enregistrées dans les registres internes du capteur. Il s'agit de nombres signés à virgules codés sur 16 ou 14 `bits`.

`Padc` et `Tadc` correspondent aux valeurs brutes de la pression et de la température mesurées par le capteur. Ces valeurs codées sur `10 bits` sont également mémorisées dans les registres internes du capteur.

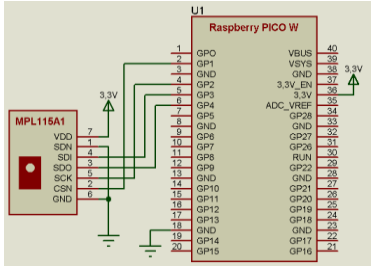
Le programme ci-dessous permet de récupérer la pression mesurée par le capteur MPL115A1 et de l'afficher sur la console :

```
from machine import SPI, Pin
import time

MPL115A1_SPI = SPI(0, baudrate = 200000, sck = Pin(2), mosi = Pin(3), miso = Pin(4))
pin_CS = Pin(1, mode = Pin.OUT)

# Lecture et calcul coefficients
pin_CS.value(0)                          # Sélection périphérique
MPL115A1_SPI.write(bytearray([0x88]))
a0Msb = int.from_bytes(MPL115A1_SPI.read(1), "little")
MPL115A1_SPI.write(bytearray([0x8A]))
a0Lsb = int.from_bytes(MPL115A1_SPI.read(1), "little")
MPL115A1_SPI.write(bytearray([0x8C]))
b1Msb = int.from_bytes(MPL115A1_SPI.read(1), "little")
MPL115A1_SPI.write(bytearray([0x8E]))
b1Lsb = int.from_bytes(MPL115A1_SPI.read(1), "little")
MPL115A1_SPI.write(bytearray([0x90]))
b2Msb = int.from_bytes(MPL115A1_SPI.read(1), "little")
MPL115A1_SPI.write(bytearray([0x92]))
b2Lsb = int.from_bytes(MPL115A1_SPI.read(1), "little")
MPL115A1_SPI.write(bytearray([0x94]))
c12Msb = int.from_bytes(MPL115A1_SPI.read(1), "little")
MPL115A1_SPI.write(bytearray([0x96]))
c12Lsb = int.from_bytes(MPL115A1_SPI.read(1), "little")
```

```
pin_CS.value(1)                         # Désélection périphérique
time.sleep_ms(3)

a0 = (a0Msb << 8 | a0Lsb) / 2**3
b1 = -(((~(b1Msb << 8 | b1Lsb)) & 0x7FFF)+1) / 2**13
b2 = -(((~(b2Msb << 8 | b2Lsb)) & 0x7FFF)+1) / 2**14
c12 = ((c12Msb << 8 |  c12Lsb) >> 2) / 2**22

while True:
    # Lire la pression et la température
    pin_CS.value(0)              # Sélection périphérique
    MPL115A1_SPI.write(bytearray([0x24])) # Lancer la conversion
    MPL115A1_SPI.write(bytearray([0x00]))
    pin_CS.value(1)              # Désélection périphérique
    time.sleep_ms(3)

    pin_CS.value(0)              # Sélection périphérique
    MPL115A1_SPI.write(bytearray([0x80]))
    padcMsb = int.from_bytes(MPL115A1_SPI.read(1), "little")
    MPL115A1_SPI.write(bytearray([0x82]))
    padcLsb = int.from_bytes(MPL115A1_SPI.read(1), "little")
    MPL115A1_SPI.write(bytearray([0x84]))
    tadcMsb = int.from_bytes(MPL115A1_SPI.read(1), "little")
    MPL115A1_SPI.write(bytearray([0x86]))
    tadcLsb = int.from_bytes(MPL115A1_SPI.read(1), "little")
    pin_CS.value(1) )            # Désélection périphérique
    time.sleep_ms(3)

    Pression = (padcMsb << 8 | padcLsb) >> 6
    Temperature = (tadcMsb << 8 | tadcLsb) >> 6

    #  Calcul de la pression compensée
    Pcomp = a0 + (b1 + c12 * Temperature) * Pression + b2 * Temperature

    # Calcul et affichage de la pression absolue
    Pabs = Pcomp * (115 - 50) / 1023.0 + 50
    Phecto = Pabs * 10;
    print("Pression : ", Phecto, " hecto Pascal")
    time.sleep(10)
```

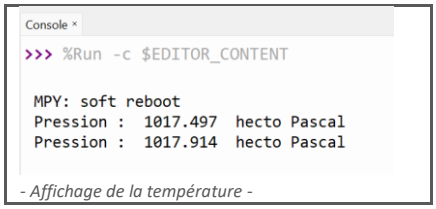


- Schéma de cablâge -



- Schéma struturel -

Après avoir exécuté le programme, la pression atmosphérique mesurée par le capteur MPL115A1 est affichée toutes les secondes sur la console.



- Affichage de la température -

8 – COMMUNICATION UART AVEC LA CARTE RASPBERRY PICO

8.1 – Interfaces UART de la carte Raspberry PICO

La carte Raspberry PICO intègre 2 ports UART : **`UART0`**, **`UART1`** indépendantes l'une de l'autre. Plusieurs broches GPIO permettent d'accéder aux lignes des interfaces SPI.

Interface	TX	RX
UART0	GPIO0, GPIO12, GPIO16	GPIO1, GPIO13, GPIO17
UART1	GPIO4, GPIO8	GPIO5, GPIO9

Les interfaces UART sont gérées par la classe **`UART`** de la librairie **`machine`**. Il faut créer un objet de type **`UART`**, sur lequel seront appliqués les différentes méthodes, en spécifiant le numéro de l'interface, la vitesse de communication (paramètre **`baudrate`**) et le numéro des broches **`Rx`** et **`Tx`**.

```
from machine import UART
nom_UART = UART(numero_Interface, baudrate = vitesse, tx = Pin(num_pin), rx = Pin(num_pin))
```

Les principales méthodes de cette librairie sont :

Méthode « any »

Cette méthode renvoie le nombre d'octets reçus sur la ligne Rx.

```
donnees = nom_UART.any()
```

Méthode « read »

Cette méthode lit les octets reçus sur la ligne Rx.

```
donnees = nom_UART.read()
```

Méthode « readline »

Cette méthode lit une ligne de caractères reçus sur la ligne Rx. Un caractère de saut de ligne termine la ligne

```
donnees = nom_UART.readline()
```

Méthode « Write »

Cette méthode écrit les octets stockés dans un objet **`donnees`** sur la ligne Rx.

```
nom_UART.write(donnees)
```

8.2 – Exemple : Communication par module Bluetooth

Dans cet exemple, la carte Raspberry PICO va permettre de transmettre des par Bluetooth via le module HC-05. La communication entre la carte Raspberry PICO et le module HC-05 sera réalisée via une interface UART.

- Module HC-05 -

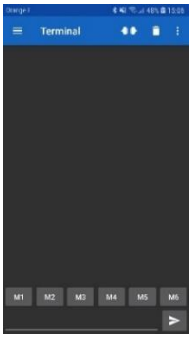
Le programme ci-dessous permet d'envoyer par Bluetooth une valeur qui s'incrémente toutes les secondes.

```
from machine import Pin, UART
import time

bluetooth = UART(1, baudrate = 9600, tx = Pin(4), rx = Pin(5))
compteur = 0

while True:
    print("Valeur du compteur: ", compteur);
    compteur = compteur + 1
    bluetooth.write("Valeur du compteur: " + str(compteur) + '\n')
    time.sleep(1)
```

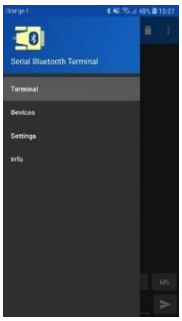
Il existe de nombreuses applications Android permettant la communication Bluetooth. Nous allons utiliser **Serial Bluetooth Terminal** .

Compiler et téléverser le programme précédent.

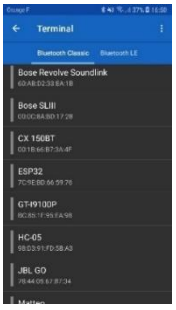
Après avoir appairé le téléphone portable avec la carte Raspberry PICO, il faut lancer l'application **Serial Bluetooth Terminal**.

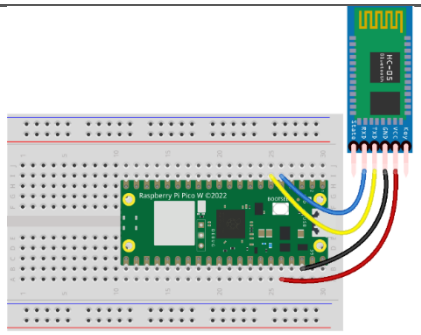


Dans le menu général choisir l'onglet « **Devices** ».

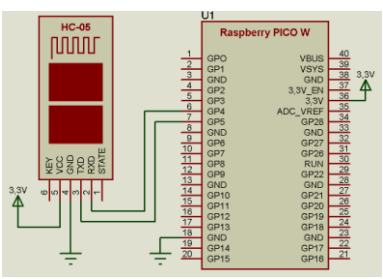


Dans la liste choisir **HC-05**.menu général choisir l'onglet « **Devices** ».





- Schéma de cablâge -



- Schéma struturel -

Les valeurs transmises par la carte Raspberry PICO via le module HC-05 s'affichent alors sur le terminal de l'application Serial Bluetooth Terminal et sur la console.

```
Console ×
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
Valeur du compteur:  0
Valeur du compteur:  1
Valeur du compteur:  2
Valeur du compteur:  3
Valeur du compteur:  4
Valeur du compteur:  5
Valeur du compteur:  6
Valeur du compteur:  7
Valeur du compteur:  8
```

- Résultats après exécution du programme -

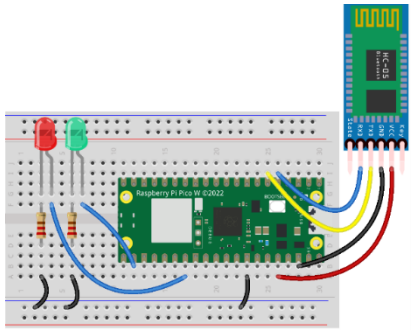
8.3 – Exemple : Réception d'une donnée par Bluetooth

Le programme ci-après permet de contrôler deux LED à partir d'une commande envoyée par un smart phone :

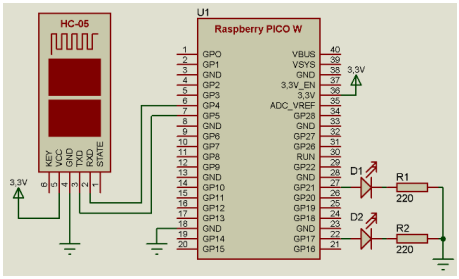
- La commande **'a'** allume uniquement la première LED
- La commande **'b'** allume uniquement la deuxième LED
- La commande **'c'** allume les deux LEDs
- La commande **'d'** éteint les deux LEDs
- La commande **'e'** change l'état des deux LEDs

```
from machine import Pin, UART
import time

bluetooth = UART(1, baudrate = 9600, tx = Pin(4), rx = Pin(5))

bluetooth_msg = ""
etatLED1 = 0
etatLED2 = 0
pin_led1 = Pin(17, mode=Pin.OUT)
pin_led2 = Pin(21, mode=Pin.OUT)

while True:
    if bluetooth.any() > 0:
        bluetooth_msg = str(bluetooth.read().strip().decode()):
        if bluetooth_msg == 'a':
            pin_led1.value(1)
            pin_led2.value(0)
            etatLED1 = 1
            etatLED2 = 0
            bluetooth.write("LED 1 allumée, LED 2 éteinte \n")
        elif bluetooth_msg == 'b':
            pin_led1.value(0)
            pin_led2.value(1)
            etatLED1 = 0
            etatLED2 = 1
            bluetooth.write("LED 1 éteinte, LED 2 allumée \n")
        elif bluetooth_msg == 'c':
            pin_led1.value(1)
            pin_led2.value(1)
            etatLED1 = 1
            etatLED2 = 1
            bluetooth.write("Les 2 LEDs allumées \n")
        elif bluetooth_msg == 'd':
            pin_led1.value(0)
```

```
            pin_led2.value(0)
            etatLED1 = 0
            etatLED2 = 0
            bluetooth.write("Les 2 LEDs éteintes \n")
        elif bluetooth_msg == 'e':
            etatLED1 = not etatLED1
            etatLED2 = not etatLED2
            pin_led1.value(etatLED1)
            pin_led2.value(etatLED2)
            bluetooth.write("Les LEDs ont changé d'état \n")
        bluetooth_msg = ""
        time.sleep(1)
```

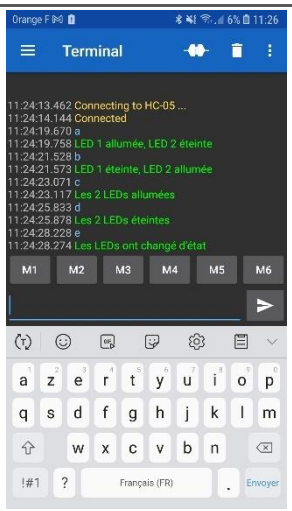


- Schéma de cablâge -



- Schéma structurel -

Après avoir téléversé le programme, il devient possible de commander à distance par Bluetooth l'allumage des 2 LEDs à l'aide de l'application Serial Bluetooth Terminal par exemple.



- Résultats après exécution du programme -