AIDE MICROPYTHON POUR RP2

Raspberry Pi Pico et Pico W, Thonny IDE,
Drivers DRV8833, VNH5019, A4988

Doc. de référence : https://docs.micropython.org/en/latest/rp2/quickref.html
Livre de référence : https://www.editions-eni.fr/livre/raspberry-pi-pico-et-pico-w-la-programmation-python-sur-microcontroleur-avec-micropython-9782409038754

S. Juhel, J. Ferry Versailles

Table des matières

Présentation Raspberry Pi Pico

1	Instructions connexion Pico	« Premiers pas » et utilisation de Thonny.
2	Gestion entrée binaire extérieure	Utilisation d'un bouton poussoir extérieur (allumage led intérieure)
3	Gestion sortie binaire	Allumage d'une led extérieure
4	Gestion entrée analogique	Utilisation d'un potentiomètre rotatif et affichage sur une courbe temporelle.
5	Gestion sortie PWM	Variation luminosité d'une led
6	Gestion sortie PWM pour servomoteur	Variation vitesse d'un servomoteur
7	Gestion des timers	Pilotage d'un moteur pas à pas
8	Sauvegarde données dans fichier	Sauvegarde de mesure d'un potentiomètre dans un fichier csv
9	Gestion des interruptions	Comptage d'appui sur un bouton indépendant du temps de cycle
10	Connexion capteur bus I²C	Communication avec une centrale de mesure gyroscopique par bus I²C
11	Installation d'une classe python	Installation d'une classe python téléchargée pour gestion périphérique
12	Connexion en mode mobilité	Pour placer votre programme sur la carte afin de la libérer du câble USB
13	Connexion en WIFI	Communication sans fil entre Thonny et la carte (uniquement avec la Pico W).
14	Mise à jour Pico	**Installation première utilisation pour votre propre carte**

- Carte E/S Pico
- Branchement hacheur DRV8833
  (pour 1 ou 2 moteurs à courant continu <1,2A)
- Branchement hacheur VNH5019
  (pour 1 moteur à courant continu <12A)
- Branchement driver A4988 (pour 1 moteur pas à pas bipolaire)

Raspberry Pi Pico Wifi

Attention FRAGILE
Ne pas mettre ses doigts dessus
Jamais plus de 3,3 V en entrée

Led verte test

Port micro USB : utile pour la communication et l'alimentation (5V)

Bouton BOOTSEL pour mise à niveau. (voir fiche 14)
Ne pas toucher

3V3 (out) : sortie Alimentation 3,3 V pour alimenter des périphériques (technologie CMOS)

Microcontrôleur RP2040

Quartz

Masse GrouND (GND)

Antenne Wifi (en option)

26 Entrées/sorties binaires GPIO (General Purpose Input Output)

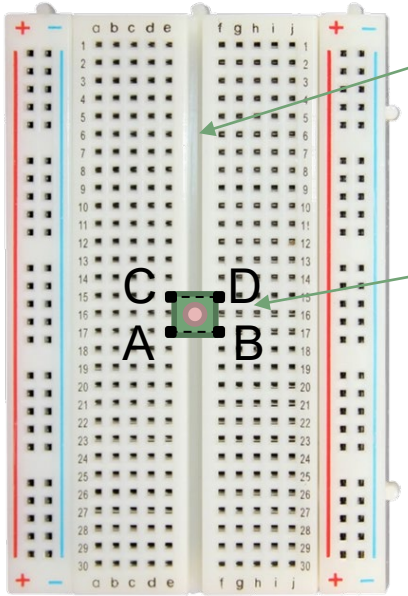
GP26, 27 et 28 sont équipées d'un convertisseur analogique/numérique 12 bits, 3,3 V max

La carte contient aussi un capteur de température.

Pour en savoir plus sur les entrées/sorties fig. Pico

1 Branchement sur une carte de prototypage (breadboard)

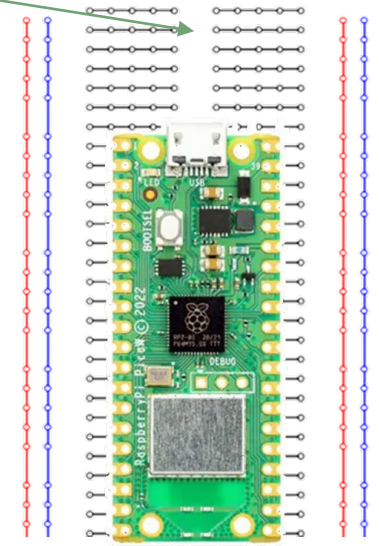
- Les cartes et les éléments électroniques seront placés à cheval de la colonne médiane d'une carte de prototypage (breadboard) qui a la structure suivante :



Colonne médiane

Bouton poussoir (notez l'emplacement des picots métalliques).
Le bouton fait le pont entre partie gauche et droite de la colonne médiane (A est relié à B et C est relié à D). Quand on appuie dessus, les lignes horizontales (AB) et (CD) sont reliées.

Une colonne rouge peut servir à partager l'alimentation 3V3 (reliée à 3V3 (out) de la pico).

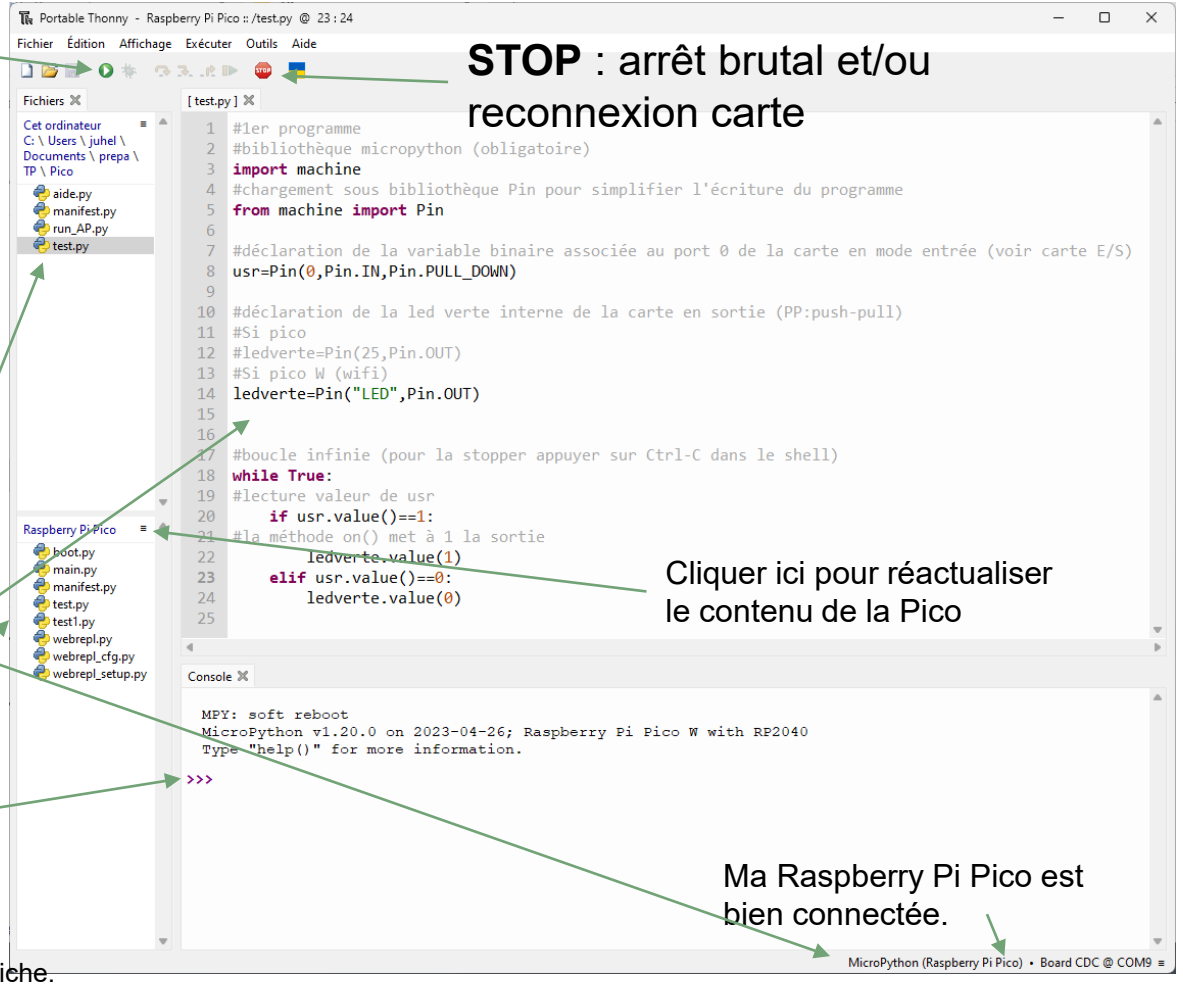


Une colonne bleue peut servir à partager la masse (reliée à GND de la pico).

1 Instructions Thonny IDE

- La carte Pico est branchée à l'ordinateur (PC) par le port USB. Ce port sert aussi à son alimentation électrique.
- Ouvrir le logiciel Thonny IDE. **Attendre quelques instants que le PC détecte la carte.**
- Cliquer sur la flèche ici.
- Si pb de connexion, cliquer sur le bouton STOP
- 3 espaces :
  - Editeur de code python
  - Accès aux fichiers PC ou Pico
  - Shell Python de la Pico.
  - Il est aussi possible de faire apparaitre des graphes.

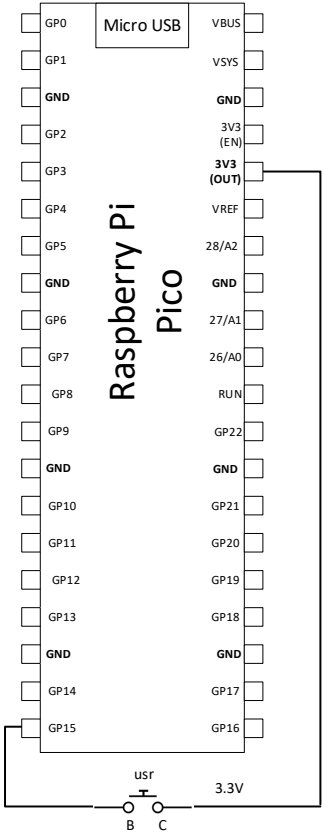
Le module micropython a été préinstallé sinon voir dernière fiche.



1 Instructions shell et lancement programme

- Si la carte n'est pas visible après branchement, cliquer sur le bouton STOP.
- Vous pouvez tester une instruction python de base dans le shell (fenêtre du bas). Les bibliothèques math et cmath sont présentes (`import math`) (ou `import cmath`) mais pas numpy.
- Pour exécuter un programme présent dans l'éditeur de Thonny, cliquer sur le bouton vert « run » (voir diapo précédente).
- **IMPORTANT : c'est bien la carte qui exécute votre code et non le PC mais votre programme est sur le PC (ou disque réseau).**
- Pour interrompre un programme, appuyer sur **Ctrl-C**. Les variables globales et les procédures éventuelles sont accessibles. (très utile en mode débug).
- Pour relancer le shell (et donc effacer la mémoire) sans déconnecter la carte (soft reboot) vous pouvez cliquer sur le bouton **STOP** (arrêt brutal) de Thonny.

2 Acquisition binaire à partir d'un bouton extérieur*



- Ici l'entrée physique choisie est GP15 (en bas à gauche). C'est elle qui recevra le signal électrique indiquant que l'on a appuyé sur le bouton extérieur nommé usr. (user)

- Pin.PULL_DOWN limite le phénomène de rebond électrique pour les boutons normalement ouverts.
- Pour lancer le programme cliquer sur la flèche verte.
- Pour stopper le programme en cours taper Ctrl-C dans le shell.

Réécrire uniquement les lignes de code.

Respecter bien les tabulations et la casse !

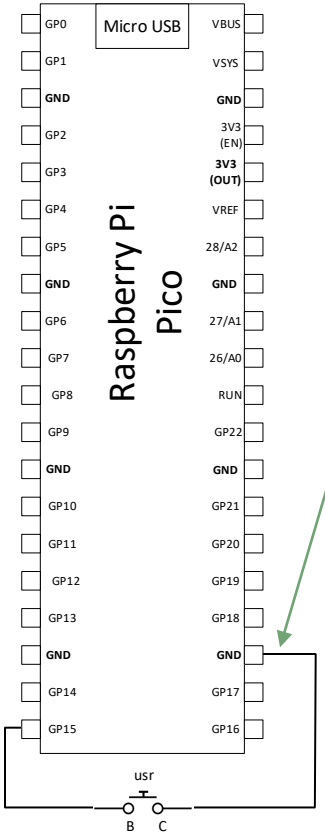
Attention aux branchements du bouton ; A et B sont toujours reliés.

```
#bibliothèque micropython (obligatoire)
import machine
#chargement sous-bibliothèque Pin pour simplifier l'écriture du programme
from machine import Pin

#déclaration de la variable binaire associée au port 15 de la carte en mode entrée (voir carte E/S)
usr=Pin(15,Pin.IN,Pin.PULL_DOWN)

#déclaration de la led verte interne de la carte en sortie
#Si pico
ledverte=Pin(25,Pin.OUT)
#Si pico W (wifi)
#ledverte=Pin("LED",Pin.OUT)

#boucle infinie (pour l'interrompre, appuyer sur Ctrl-C dans le shell)
while True:
#lecture valeur de usr
    if usr.value()==1:
#la méthode value(1) met à 1 la sortie
        ledverte.value(1)
    elif usr.value()==0:
        ledverte.value(0)
```

Petite (et dernière) exception

* Ceci est le câblage pédagogique utilisé dans cette aide, le câblage pro est page suivante.

2 Acquisition binaire à partir d'un bouton extérieur (pro) 2/2



- L'alimentation 3,3 V est gérée en interne par la carte (ce qui évite de manipuler de la tension dans le câblage avec des risques de courts-circuits préjudiciables).

- Le bouton est branché sur la masse (GND).
- Pin.PULL_UP est utilisé à la place de PULL_DOWN
- Le bouton se comporte comme un normalement fermé : la carte le voit à 0 logique quand il est appuyé.

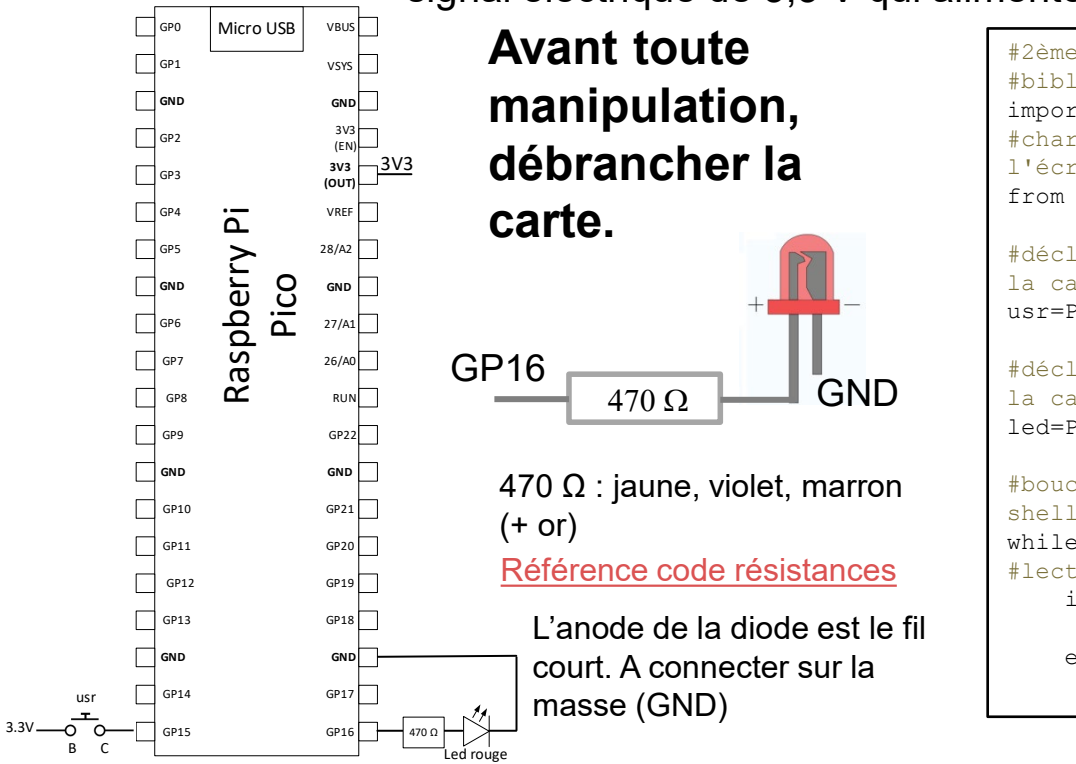
```
#bibliothèque micropython (obligatoire)
import machine
#chargement sous-bibliothèque Pin pour simplifier l'écriture du programme
from machine import Pin

#déclaration de la variable binaire associée au port 15 de la carte en mode entrée (voir carte E/S)
usr=Pin(15,Pin.IN,Pin.PULL_UP)

#déclaration de la led verte interne de la carte en sortie
#Si pico
ledverte=Pin(25,Pin.OUT)
#Si pico W (wifi)
#ledverte=Pin("LED",Pin.OUT)

#boucle infinie (pour l'interrompre, appuyer sur Ctrl-C dans le shell)
while True:
#lecture valeur de usr
    if usr.value()==0:
#la méthode value(1) met à 1 la sortie
        ledverte.value(1)
    elif usr.value()==1:
        ledverte.value(0)
```

Petite (et dernière) exception

Attention aux branchements du bouton ; A et B sont toujours reliés.

3 Sortie binaire à partir d'un bouton extérieur

- Ici la sortie physique choisie est GP16 (en bas à droite). La carte fournira un signal électrique de 3,3 V qui alimentera la led à la demande.



Avant toute manipulation, débrancher la carte.

470 Ω : jaune, violet, marron (+ or)

Référence code résistances

L'anode de la diode est le fil court. A connecter sur la masse (GND)

```
#2ème programme
#bibliothèque micropython (obligatoire)
import machine
#chargement sous bibliothèque Pin pour simplifier
l'écriture du programme
from machine import Pin

#déclaration de la variable binaire associée au port 15 de
la carte en mode entrée (voir carte E/S)
usr=Pin(15,Pin.IN,Pin.PULL_DOWN)

#déclaration de la variable binaire associée au port 16 de
la carte en mode sortie (voir carte E/S)
led=Pin(16,Pin.OUT)

#boucle infinie (pour la stopper appuyer sur Ctrl-C dans le
shell)
while True:
#lecture valeur de usr
    if usr.value()==1:
        led.value(1) #la led s'allume
    elif usr.value()==0:
        led.value(0) #la led s'éteint
```

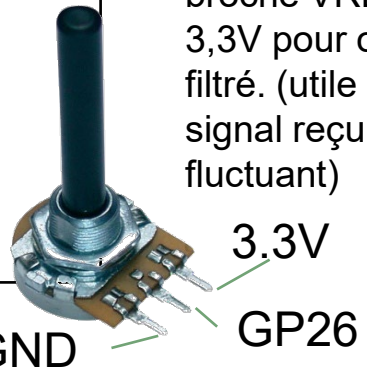
4 Acquisition entrée analogique et affichage grapheur (1/2)

- Acquisition sur 12 bits mais affichage sur 16 (0-65535) pour les entrées associées au CAN (Convertisseur Analogique Numérique).
- Seuls les ports GP26-27-28 sont équipés de CAN (ADC en anglais : Analog Digital Converter).
- Attention 3,3 V maximum

```
#3ème programme
#bibliothèque micropython (obligatoire)
import machine
import time
#chargement sous bibliothèque Pin pour simplifier l'écriture du programme
from machine import Pin

#déclaration de la variable associée au port GP26 en mode analogique (voir carte E/S)
pot=machine.ADC(Pin(26))

#boucle infinie (pour la stopper appuyer sur Ctrl-C dans le shell)
while True:
#lecture valeur de usr
    val=pot.read_u16() #pour rappel la résolution n'est que de 12 bits
    print(val) #affichage écran de val
    time.sleep(1) #attente de 1 seconde.
```

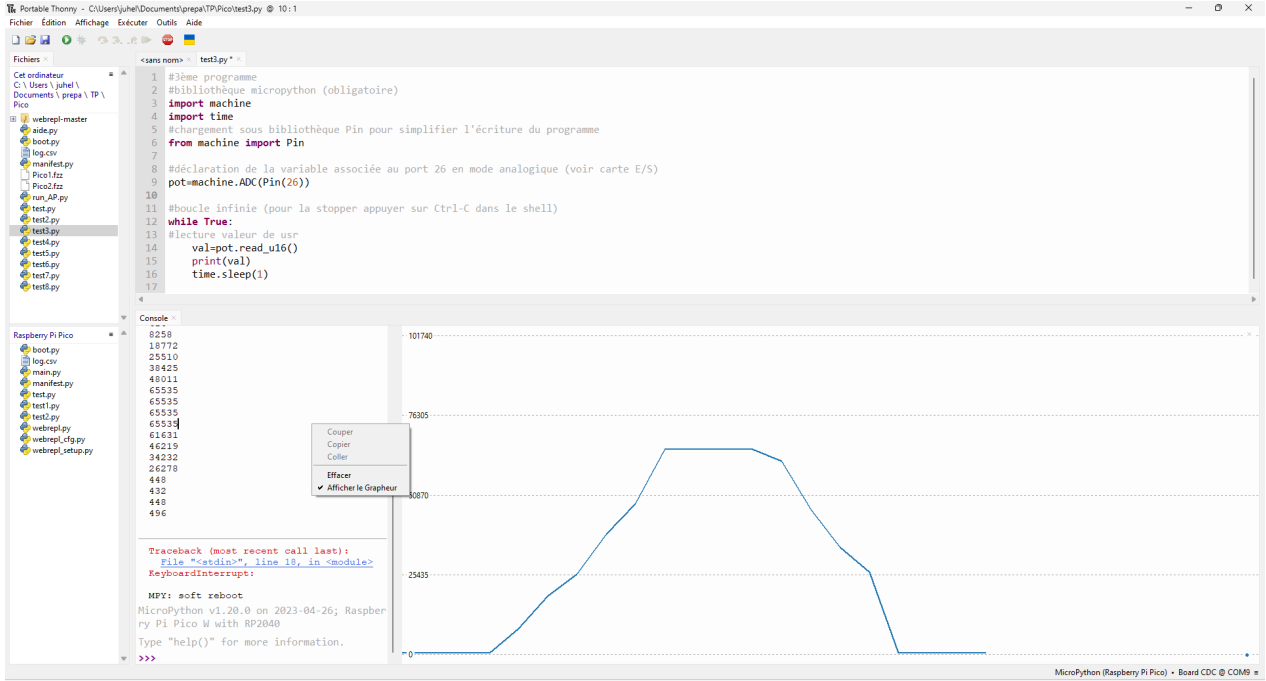
Respecter les tabulations et la casse !

- Pour tester ce programme il faut brancher un potentiomètre sur GP26, et le 3.3V (out) de la carte d'une part et sur la masse (GND) d'autre part.
- Il est possible d'utiliser la broche VREF à la place du 3,3V pour obtenir 3,3 V filtré. (utile seulement si le signal reçu est trop fluctuant)



4 Acquisition entrée analogique & affichage grapheur (2/2)

- Les valeurs affichées dans le shell (méthode print) peuvent être représentées en fonction du temps dans le grapheur. (menu contextuel Afficher le Grapheur)



5 Sortie PWM (Pulse Width Modulation)

- Permet de fournir un signal créneau dont la largeur est réglable de 0 à 100%. A 100% le signal est continu à 3.3V.
- Les fréquences sont de 8Hz à 62,5 MHz.
- La largeur d'impulsion doit être fournie sur 16 bits (soit de 0 à 65535)
- Il y a 8 sorties PWM avec 2 canaux (nommés slices) chacun. Ex : GP0 et GP1 travailleront à la même fréquence et seront synchrones mais pas forcément à la même largeur d'impulsion. (voir carte E/S)
- Pour en savoir + faire le TP ventilo

Exemple de programme qui permet de modifier l'intensité lumineuse de la led branchée sur le port 16.
***Voir pages précédentes** pour le branchement de la led et du potentiomètre*

Respecter les tabulations et la casse !

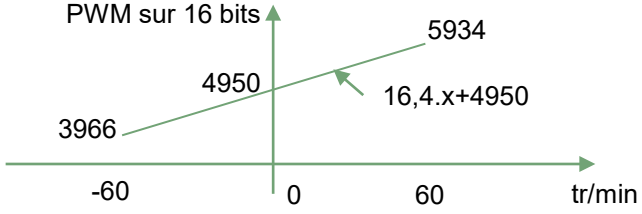
```
#4ème programme
#bibliothèque micropython (obligatoire)
import machine
from machine import Pin

#déclaration de la variable associée au port 16 (voir carte E/S)
#toutes les sorties peuvent émettre des signaux créneaux de type MLI
(PWM)
mli=machine.PWM(Pin(16))
mli.freq(50) #fréquence de 50Hz

#déclaration de la variable associée au port 26 en mode analogique
(voir carte E/S) (voir programme 3)
pot=machine.ADC(Pin(26))

#boucle infinie (pour la stopper appuyer sur Ctrl-C dans le shell)
while True:
    #lecture valeur de usr
    val=pot.read_u16()
    #envoi de la largeur d'impulsion sur 16 bits de 0-65535
    mli.duty_u16(int(val))
    #envoi en % de largeur d'impulsion avec limp compris entre 0 et 1.
    #mli.duty_u16(int(limp*65535))
```

MLI : Modulation par Largeur d'Impulsions = PWM

6 Sortie PWM (Pulse Width Modulation) ex : commande servomoteur

MLI : Modulation par Largeur d'Impulsions = PWM

- Commande d'un servomoteur de vitesse de modélisme modèle FS5106R de Fitec.
- Le document constructeur impose un signal de commande de type MLI à 50 Hz soit une période de 20 ms. La vitesse nulle est obtenue avec un créneau de 1,5 ms soit 7,5% de la période (soit 4950 environ, 7,5% de 65535 –pwm sur 16 bits-)). La vitesse max en sens trigo (vue de face) pour 1,8 ms (soit 9% de la période – 5934-) et 6% (1,2 ms) pour la vitesse max sens horaire.
- Ecriture d'une procédure servo paramétrée par val en tour/min. val est positif dans le sens trigonométrique. Donc dans la console de Thonny, il faut taper servo(30) pour obtenir une vitesse de 30 tr/min.
- Pour 5V d'alimentation la vitesse max est approximativement de 60 tr/min. (à affiner).




```
#bibliothèque micropython (obligatoire)
import machine
from machine import Pin

#déclaration de la variable associée au port 16 (voir carte E/S)
#toutes les sorties peuvent émettre des signaux créneaux de type
MLI (PWM)
#ici la GP16
mli=machine.PWM(Pin(16))
mli.freq(50) #fréquence de 50Hz

def servo(val) :
    if -1<val and val<1:
        pwm=4950 #correspond à 7,5% environ de 65535 (arrêt)
    else :
        pwm=int(val*16.4+4950) #tout est proportion de 65535
    #print(pwm)
    mli.duty_u16(pwm)
    return
```

Pour les servomoteurs de position, même principe mais la largeur d'impulsions à fournir impose une position angulaire (à tester).

Branchement :

- Fil rouge sur 5V (alimentation ext.)
- Fil marron sur masse alimentation ext.
- Fil orange + résistance 10kΩ sur GP16 de la Pico (sortie pwm choisie)

```
import machine,time
from machine import Pin,Timer #(modules à importer)

#déclaration des ports de sorties (voir câblage A4988)
dir_pin = Pin(20, Pin.OUT)
step_pin = Pin(21, Pin.OUT)
enable_pin = Pin(22, Pin.OUT)
step=0

#fonction de pilotage du driver A4988 (t est un paramètre obligatoire)
def commande_moteur(t):
  global step          #variable globale car utile dans d'autres boucles du programme
  step+=1              #incrémentation de 1 de step
  step_pin.high()
#envoi du signal créneau de 2 µs sur le port step de l'A4988 équivalent step_pin.value(1)
  time.sleep_us(2) #créneau de largeur 2µs
  step_pin.low()

#données constructeur moteur
N=200                  #nbre pas/tour (donnée constructeur)
r=1/(5.18)             #rapport de réduction (donnée constructeur)
#consignes utilisateur
pas=1036               #nbre pas moteur désirés (ici 1 tour sortie réducteur (N/r))
vit_rot=10             #vitesse [tr/min] en sortie réducteur à modifier (ordre)
frequence=(N*vit_rot)/(60*r)
dir_pin.low()          # high pour le sens antitrigo de rotation (vue de face de l'axe)

tim=Timer(mode=Timer.PERIODIC, freq=frequence, callback=commande_moteur)
#Ce timer interne ne sera pas utilisé avec une sortie mais est associé à la procédure callback
enable_pin.low()     #deblocage moteur

while True :
  time.sleep(0.1)
  print(step)
  if step>=pas:
    enable_pin.high() # arrêt du moteur
    tim.deinit()       #arret du timer tim
    break
```

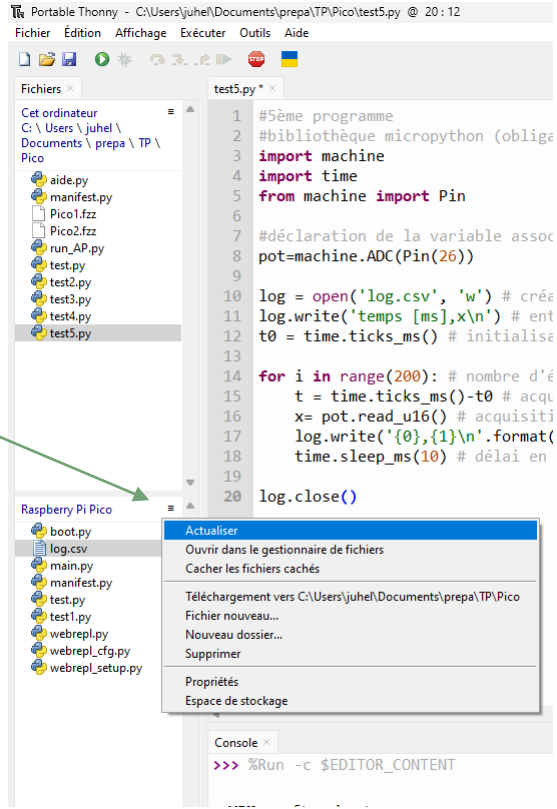
7 Gestion des timers ex : pilotage d'un moteur pas à pas

- La classe Timer de « machine » contient des méthodes capables de générer des signaux à fréquences imposées (indépendamment du programme principal) (comme le signal PWM voir fiche précédente).
- Pour l'exemple, la durée des impulsions est de 2µs.
- Le moteur pas à pas choisi est bipolaire (2 bobines à commander – 4 fils). (ex : nema16 avec présence d'un réducteur optionnel).
- Pour simplifier programme et **câblage** un driver spécialisé est utilisé : le Pololu A4988
- Ici le timer n'est pas lié à une sortie particulière mais à une fonction (callback).

8 Enregistrement d'un fichier de mesure

- Instructions permettant de créer un fichier texte (.csv) qui pourra être manipuler par du code python ou un tableur.
- Dans Thonny, il faut penser à actualiser avant d'ouvrir le fichier (avec un éditeur de texte ou tableur voire Thonny)

```
#5ème programme
import machine,time
from machine import Pin

#déclaration de la variable associée au port 26 en mode analogique (voir carte E/S)
pot=machine.ADC(Pin(26))

# création du fichier log.csv ou écrasement si déjà existant dans la mémoire de la Pico
log = open('log.csv', 'w')
# écriture de l'entête du fichier csv de 2 colonnes ici
log.write('temps [ms],x\n') #\n correspond à un retour à la ligne

# initialisation temporelle en ms
t0 = time.ticks_ms()

for i in range(200): # 200 : nombre d'échantillons désirés
    t = time.ticks_diff(time.ticks_ms(), t0) # acquisition du temps depuis t0
    x= pot.read_u16() # acquisition des données de pot (voir 3ème programme)
    log.write('{0},{1}\n'.format(t,x)) # écriture des données dans le fichier log.csv
    time.sleep_ms(10) # pause en millisecondes (10 ici) approximativement

log.close() # fermeture du fichier pour manipulation
```



Respecter bien les tabulations et la casse !

9 Gestion des interruptions
ex : comptage d'une action sur bouton

```
#6ème programme
import machine,time
from machine import Pin

#déclaration de la variable binaire associée au port 15 de la carte en mode entrée (voir carte E/S)
usr=Pin(15,Pin.IN,Pin.PULL_DOWN)

#déclaration de variables internes
compt=0
t0 = time.ticks_ms() #prise du temps interne

#définition de la fonction qui sera appelée par l'interruption (le paramètre ref est obligatoire)
def callback_usr(ref):
    global compt #compt est une variable globale donc visible dans tout le programme
    compt=compt+1 #la fonction d'interruption doit être la plus rapide possible
    return

#création de l'interruption sur l'entrée usr
usr.irq(callback_usr,trigger=Pin.IRQ_RISING) #au front montant (rising) du signal reçu par usr la fonction callback_usr est lancée en priorité.

#boucle principale infinie (pour la stopper appuyer sur Ctrl-C dans le shell)
while True:
    if time.ticks_diff(time.ticks_ms(), t0)>1000 : #affichage de compt toutes les secondes
        t0 = time.ticks_ms()
        print(compt)
```

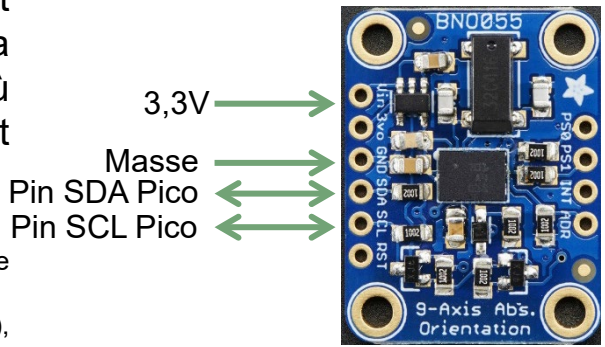
- Permet de lancer une fonction sur occurrence d'un signal binaire d'un port input (ici usr) indépendamment du temps de cycle de votre boucle principale.
- Il est conseillé de limiter la taille des instructions dans la fonction appelée.
- Si votre capteur est ultra rapide ou votre bouton de mauvaise qualité, il se peut que la carte détecte des fronts fantômes (rebonds électriques). Il faut alors ajouter un test dans la fonction callback_usr.

10 Connexion d'une centrale gyroscopique par bus I²C (1/2)

- Mise en place d'un bus de communication de type I²C pour faire communiquer la carte avec un périphérique ou une autre carte. (exemple centrale inertielle BNO055)
- Branchement : SDA Pico vers SDA périphériques ; SCL Pico vers SCL périphériques (s'il y en a plusieurs ils sont donc en //); Il faut bien entendu alimenter le périphérique et partager sa masse avec la Pico.
- On choisit ici le bus I²C N°1 de la Pico donc SCL sur GP7 et SDA sur GP6.(voir carte E/S Pico)
- Dans REPL (le shell), on écrit
  - >>>from machine import I2C
  - >>>i2c = I2C(1) #pour ouvrir le bus 1 avec la Pico en maître (par défaut)
  - >>>addrs=[hex(x) for x in i2c.scan()] #permet de connaître les adresses en hexa des esclaves branchés sur le bus ouvert (i2c) (pour notre exemple le BNO055, l'adresse imposée est 0x28)

SDA : Serial Data Line
SCL : Serial Clock Line

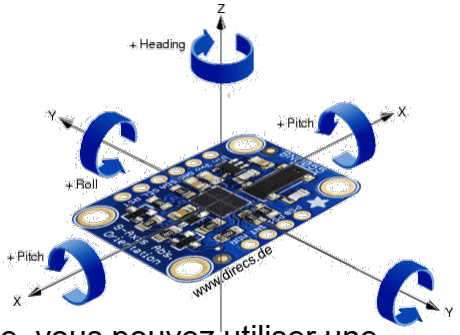
La carte Pico est donc configurée en tant que maître : elle seule peut engendrer une conversation avec les périphériques esclaves. Elle devra préciser l'adresse du périphérique, l'adresse du registre du périphérique où l'on souhaite écrire ou lire et la taille du mot qu'elle envoie ou qu'elle doit recevoir. (voir page suivante)



Si problème de communication, il faut placer une résistance de 1000 Ω entre SDA et le 3,3 V et une entre SCL et 3,3 V.
Pour valider un câblage, taper i2c et la carte renvoie : I2C(0, freq=399361, scl=5, sda=4, timeout=50000), c'est-à-dire que le I2C(0) est branché sur GP5 et GP4. (ici I2C(0) est choisi comme ex.)

10 Connexion d'une centrale gyroscopique par bus I²C (2/2)

- Lecture température sur 8 bits (1 octet)
- Lecture angle de roulis relatif (roll) sur 2 octets (16 bits).



Pour faire simple, vous pouvez utiliser une classe toute faite (voir fiches suivantes).

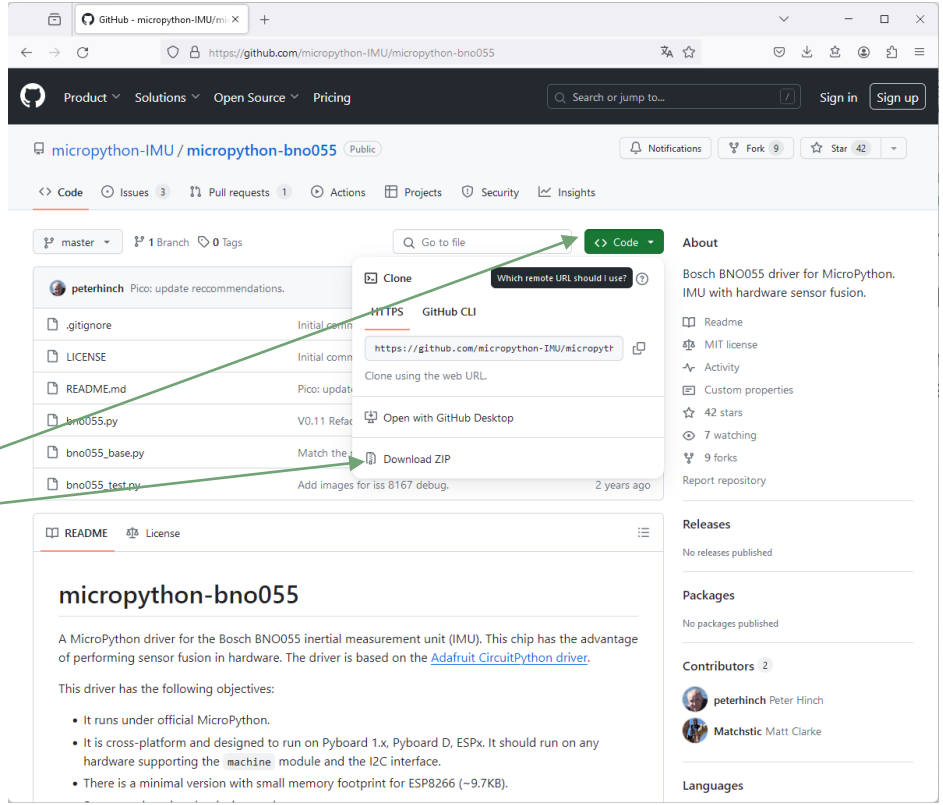
Exemple pour le BNO055 à adapter à vos périphériques

```
#7ème programme
import machine,time
from machine import I2C
i2c = I2C(1) #ouverture bus 1 en tant que maître
#on écrit l'octet x0c au format byte, de taille 8 bits (à préciser) sur
l'esclave 0x28 à l'adresse interne (0x3d) : on active la centrale (voir
doc constructeur)
i2c.writeto_mem(0x28,0x3d,b'\x0c',addrsize=8) #i2c.writeto_mem(addr,
reg, msg)

while True:
#lecture 1 octet de 0x28 à l'adresse 0x34 (mesure température en °C
pour le BNO055 - voir doc constructeur)
    temp = i2c.readfrom_mem(0x28,0x34,1)
    print('Température =',int(temp[0]))
    time.sleep_ms(100)  #attente de 100 ms - optionnel
#lecture des angles de Tait-Bryan relatifs (notés Euler dans la doc),
ici l'angle de roulis est lu sur 2 octets à partir de l'adresse 0x1c du
capteur
    TBroulis=i2c.readfrom_mem(0x28,0x1c,2)
    roulis =(int(TBroulis[1])*256+int(TBroulis[0])) #transfomation en
entier base 10
    #résultat fourni sur 2 octets mais sur 12 bits signés (le 13ème bit
est à 1 si négatif)
    if roulis>32768 : #prise en compte du 1 sur le 13ème bit
        roulis=-65535+roulis #décalage pour obtenir la valeur négative
(car du 13 au 16ème bit le gyro renvoie des 1)
    print(roulis/16) #/16 imposé par le constructeur pour obtenir un
résultat en°.
```

11 Gestion d'une classe python : Utilisation d'une centrale gyroscopique par bus I²C avec une classe fournie. (1/3)

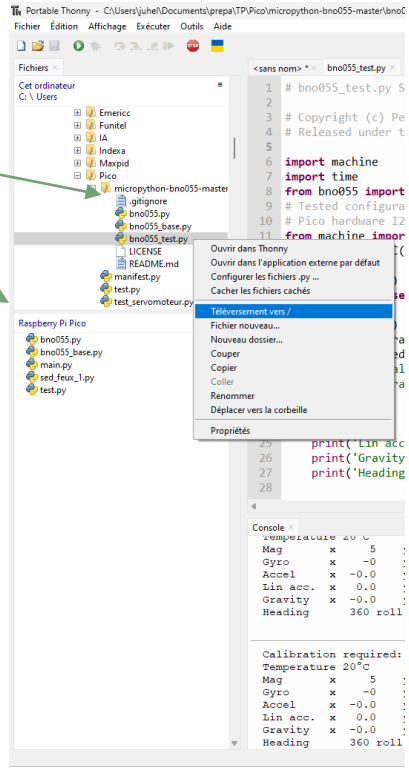
Pour utiliser des fonctions toutes faites de la communauté :

- Taper sur un moteur de recherche les mots clés « micropython » et le nom du périphériques (ici BNO055).
- Choisir un site de référence (ici github.com)
- Télécharger les fichiers puis décompresser-les dans votre répertoire de travail.



11 Gestion d'une classe python : Utilisation d'une centrale gyroscopique par bus I²C avec une classe fournie. (2/3)

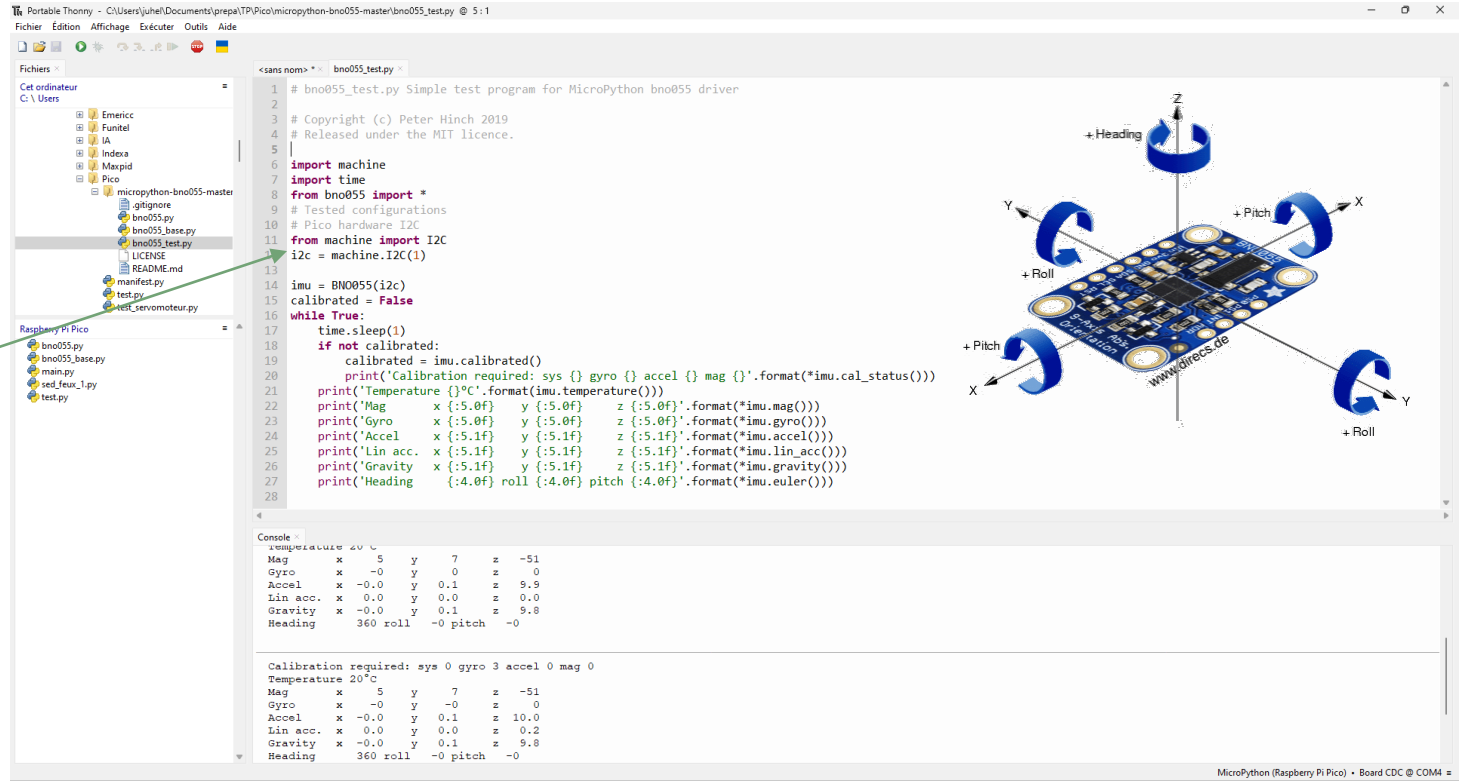
- Dans Thonny, téléverser les fichiers *.py contenant les classes du périphériques (ici bno55.py et bno55_base.py) dans la mémoire de la Pico.
- Brancher le périphérique et tester sa connexion I2C (voir fiches précédentes connexion I2C)
- Vous pouvez tester avec le code test fourni. (voir page suivante)



Exemple pour le BNO055 à adapter à vos périphériques

11 Gestion d'une classe python : Utilisation d'une centrale gyroscopique par bus I²C avec une classe fournie. (3/3)

Test du BNO055 avec le driver fourni de gihtub. (une petite adaptation sur la déclaration de l'I2C a du être écrite).



Exemple pour le BNO055 à adapter à vos périphériques

12 Instructions shell et lancement programme en **mode mobilité** (pour robot autonome par ex.)

- Pour utiliser votre Pico en mode non connectée par l'USB (pour la robotique par ex.), suivre les instructions suivantes :
- **1.** copier votre fichier .py sur la carte en cliquant sur Enregistrer sous… du menu Fichier. (et choisir Raspberry Pi Pico)
- **2.** modifier le fichier « main.py » situé sur la carte Pico (s'il n'est pas présent, le créer) en plaçant cette ligne :
  - `exec(open('test.py').read())`
  - test.py étant bien entendu le nom du fichier python que vous voulez lancer. Attention aux guillemets.

main.py est le programme lu au lancement de la carte. Attention à son contenu !

Brancher la carte avec coupleur piles avec port mini usb. Attention, le PC n'a plus de lien avec la carte. La carte est alimentée par des piles branchées sur le port USB (5,5 V max). Si vous voulez effectuer un soft reboot sans déconnecter l'USB d'alimentation, il faut brancher la masse sur l'entrée RUN (située entre GP22 et GP26 à droite). Alors main.py est exécuté à nouveau.

Tout ceci est inutile si vous êtes équipé d'une pico W (avec module WI-FI) voir fiche suivante.

13 Connexion WIFI en point d'accès (Pico W seulement) (1/2)

- Vous pouvez faire apparaitre le shell python de la Pico dans Thonny connecté en WIFI avec votre Pico W. La carte est configurée en point d'accès. Le wifi s'effectue directement entre la carte et le pc.
- Attention, seuls les programmes enregistrés dans la mémoire de la carte fonctionneront. Si vous voulez les modifier ils faut les ouvrir dans Thonny et les sauvegarder sur la carte.

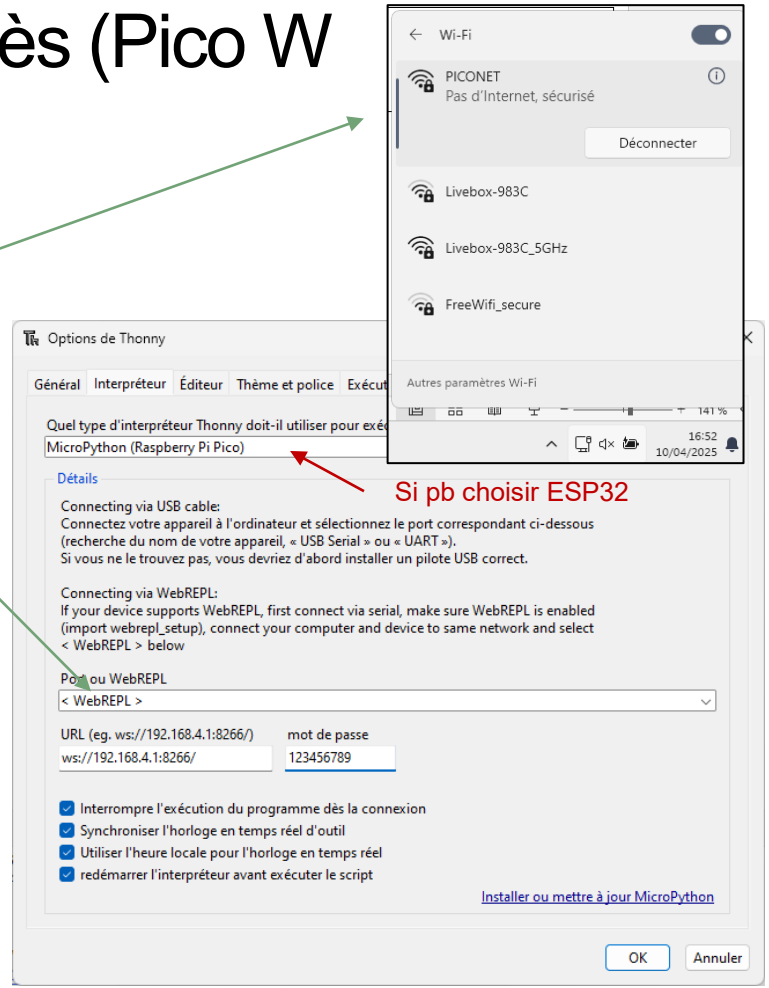
- **Etapes initiales effectuées une seule fois**
- **Etape 0.1** : mettre ses lignes dans le fichier boot.py de la carte avec Thonny et débrancher puis rebrancher la carte. Si le boot.py n'existe pas, il faut juste le créer.
- **Etape 0.2** : dans le shell taper « `import webrepl_setup` » et répondre aux questions. (Pour faire simple garder le mot de passe. Il sera placé dans un fichier webrepl_cfg.py sur la carte).
- Noter l'adresse IP qui apparait : 192.168.1.4 (par exemple)

```
#Wifi point d accès AP
from rp2 import country
import network
import socket
country('FR')
# from network import *
ap=network.WLAN(network.AP_IF)
ap.config(essid='PICONET',password='123456789')
ap.active(True)
print(ap.ifconfig())

import webrepl
webrepl.start()
```

```
Console
MicroPython v1.24.1 on 2024-11-29; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>

MPY: soft reboot
('192.168.4.1', '255.255.255.0', '192.168.4.1', '0.0.0.0')
WebREPL server started on http://192.168.4.1:8266/
Started webrepl in normal mode
MicroPython v1.24.1 on 2024-11-29; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>
```

13 Connexion WIFI en point d'accès (Pico W seulement) (2/2)

- **Etapes à chaque connexion**
- **Etape 1** : Débranchez votre carte de l'USB et alimentez-là via une batterie sur le port USB ou sur VSYS (<5 V) (voir fiche 12).
- **Etape 2** : Sur le PC, connectez-vous sur le réseau WI-FI PICONET. (parfois un délai de plusieurs minutes est nécessaire)
- **Etape 3** : Dans Thonny, aller au menu « Exécuter » puis « Configurer l'interpréteur… »
- **Etape 4** : Choisir <WebREPL> par ws://192.168.4.1:8266/ (IP noté à l'étape 0.2 et le mot de passe choisi) puis cliquer sur OK.
- Le shell apparait avec WebREPL connected écrit.
- Pour lancer un programme écrit sur la carte, taper `exec(open(« test.py »).read())` où test.py est le nom de votre programme à lancer.
- Pour écrire sur la carte menu « Fichier » puis « Enregister sous » puis carte Pico.
- Utiliser Ctrl-C pour arrêter le programme et accéder aux variables ou constantes en mémoire.
- Pour modifier votre programme, ouvrir celui qui est dans la Pico, modifiez-le et sauvegarder dans la Pico. **Ne pas utiliser le bouton flèche verte** (run de Thonny)



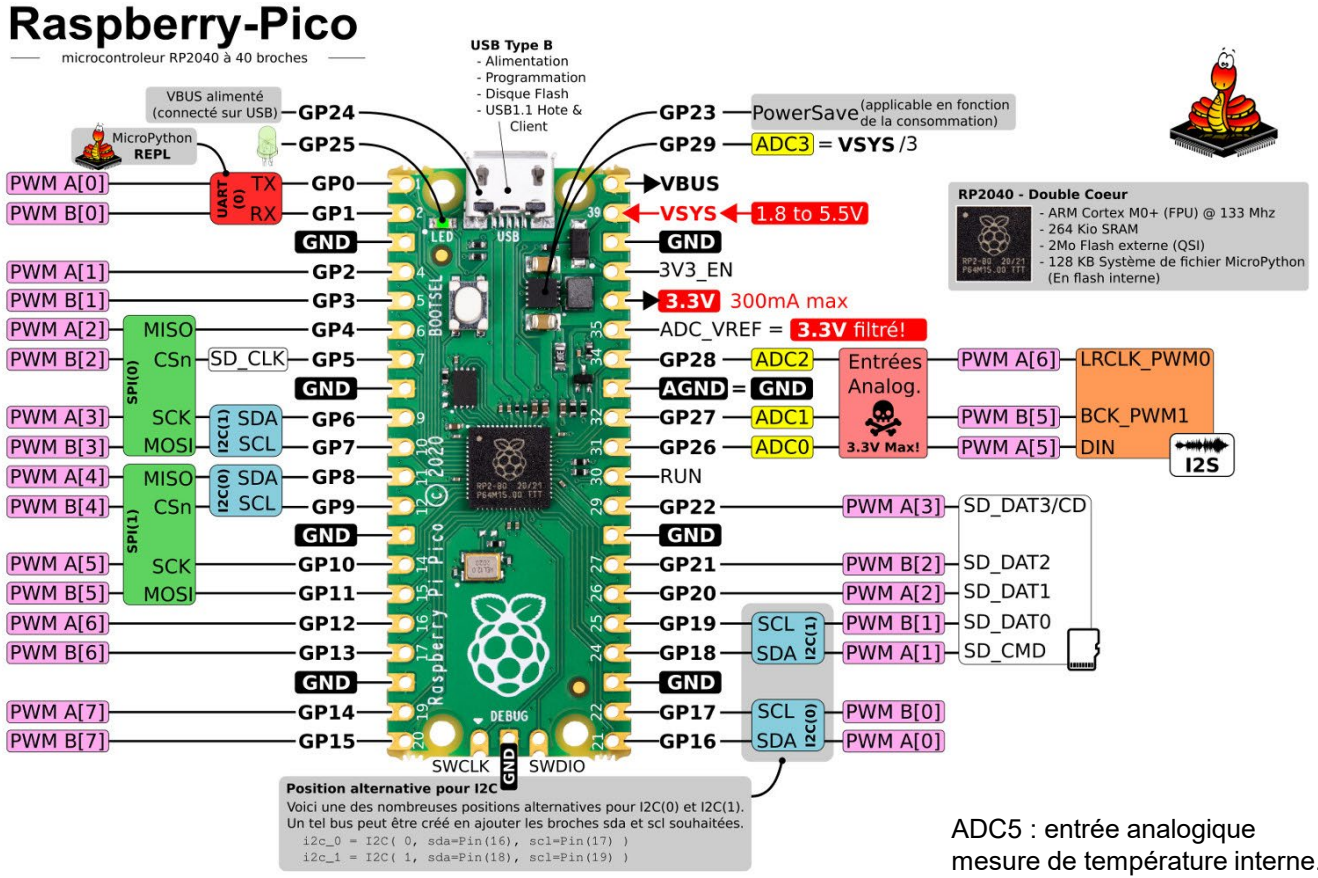
14 Mise à jour de la Pico (obligatoire au premier branchement de votre propre carte)

Le processus est très simple : il faut brancher la carte Pico sur le port USB de votre PC **tout en maintenant** appuyé le bouton BOOTSEL (le seul sur la carte).
La carte apparait comme un disque externe.
Il suffit de glisser le fichier rp2XXXX.uf2 sur la carte. (voir le lien suivant)
Ejecter la carte et rebrancher-là.
Détails ici :
https://www.raspberrypi.com/documentation/microcontrollers/micropython.html

Carte E/S Pico

Pour une alimentation autonome, privilégier un coupleur de piles à port mini-usb.

IMPORTANT : ne jamais brancher directement une sortie sur la masse !

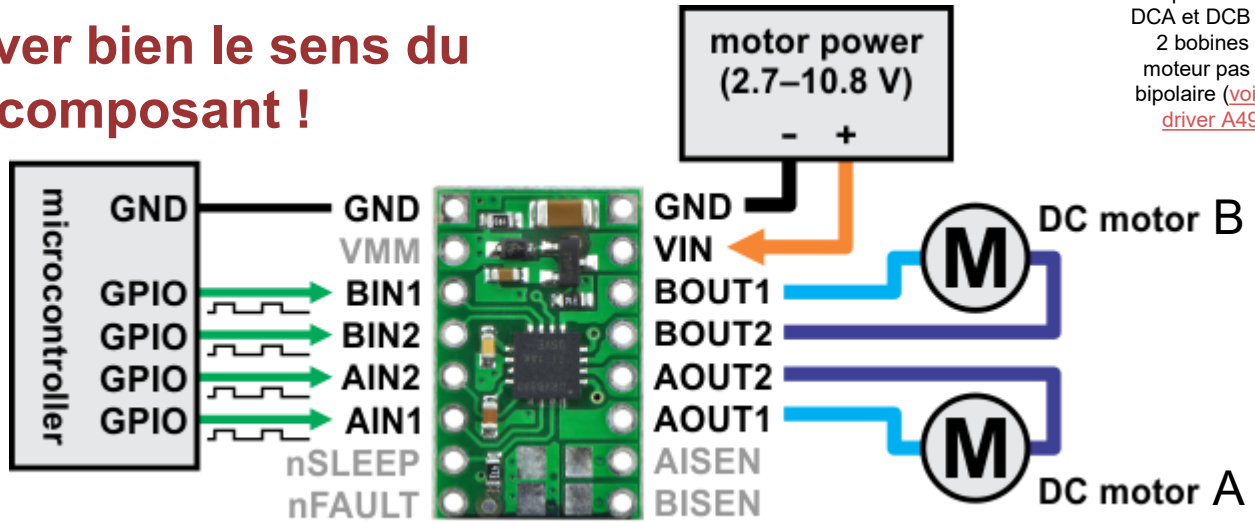


ADC5 : entrée analogique mesure de température interne.

Connexion hacheur Pololu DRV8833

1,2 A en continu

Observer bien le sens du composant !



On peut remplacer DCA et DCB par les 2 bobines d'un moteur pas à pas bipolaire (voir page driver A4988)

Fréq PWM<50kHz

Ce hacheur possède sa propre régulation de tension. Inutile donc de le relier au 3,3V de la carte microcontroleur.
Fast decay : freinage par « roue libre » freinage lent, PWM à 0 pour couper.
Slow decay : freinage par court-circuit du moteur, freinage rapide, PWM à 100 pour arrêter.

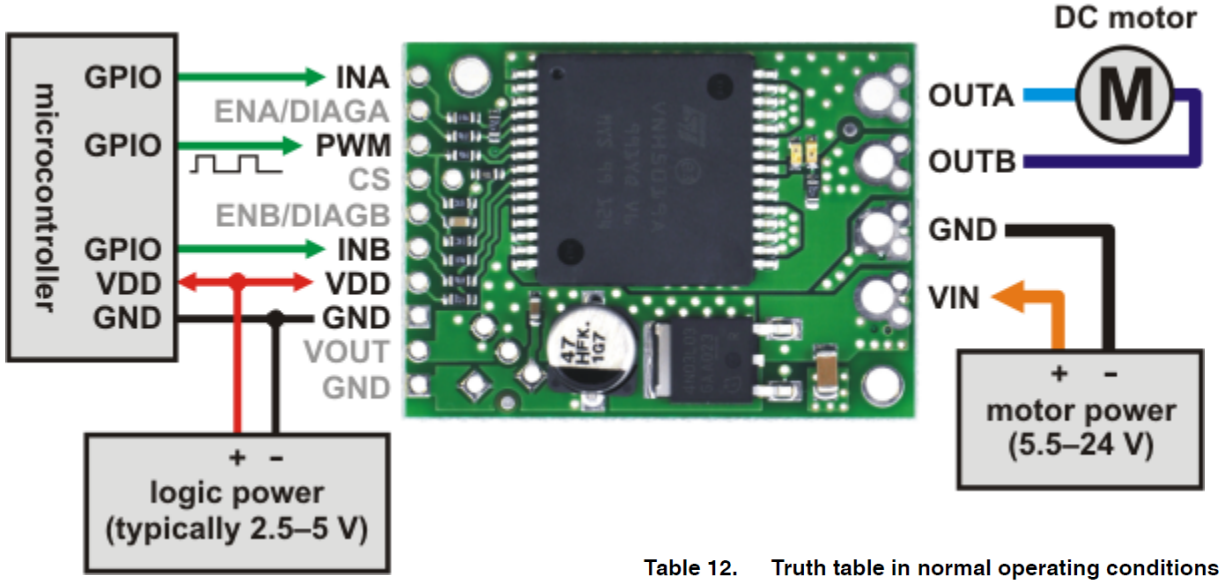
Table 2. PWM Control of Motor Speed

xIN1	xIN2	FUNCTION
PWM	0	Forward PWM, fast decay
1	PWM	Forward PWM, slow decay
0	PWM	Reverse PWM, fast decay
PWM	1	Reverse PWM, slow decay

Connexion hacheur Pololu VNH5019

12A en continu

VDD=3,3V



Important
Freq PWM<20kHz

Table 12. Truth table in normal operating conditions

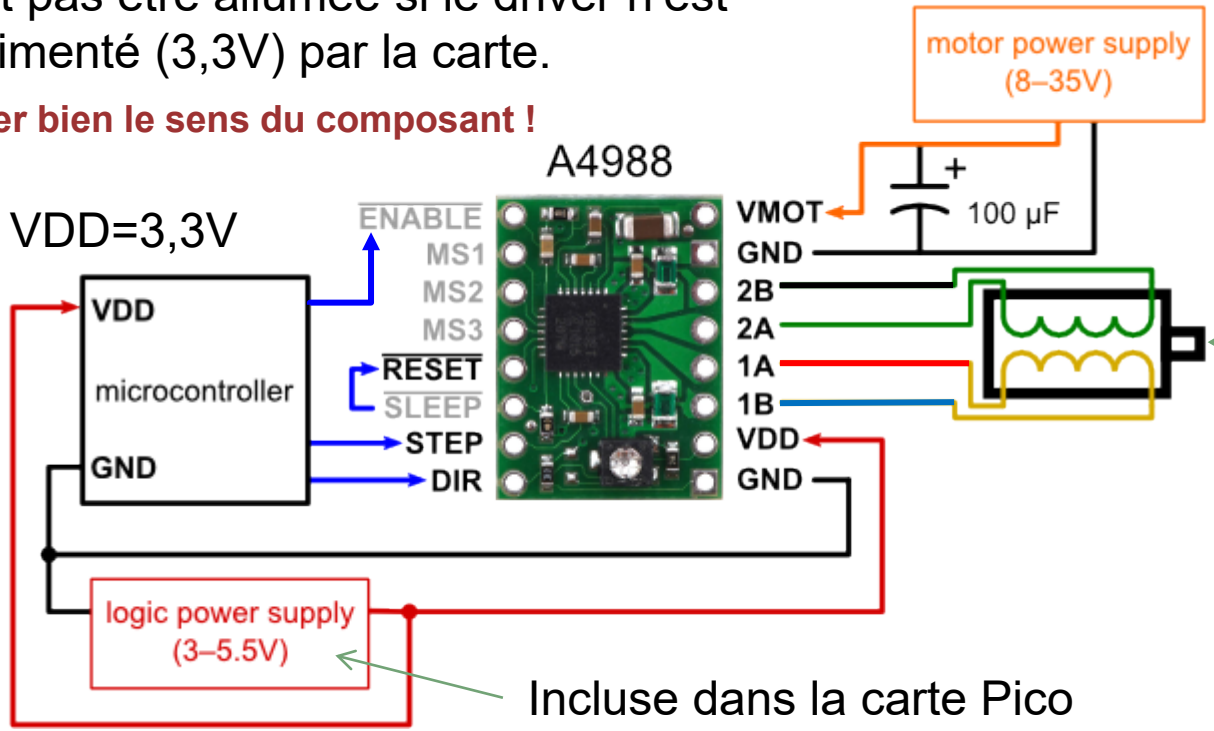
IN_A	IN_B	$DIAG_A/EN_A$	$DIAG_B/EN_B$	OUT_A	OUT_B	CS (V_{CSD} = 0 V)	Operating mode
1	1	1	1	H	H	High imp.	Brake to V_{CC}
1	0	1	1	H	L	$I_{SENSE} = I_{OUT}/K$	Clockwise (CW)
0	1	1	1	L	H	$I_{SENSE} = I_{OUT}/K$	Counterclockwise (CCW)
0	0	1	1	L	L	High imp.	Brake to GND

Connexion driver moteur pas à pas Pololu A4988

IMPORTANT : l'alimentation de puissance ne doit pas être allumée si le driver n'est pas alimenté (3,3V) par la carte.

Observer bien le sens du composant !

Si la pastille argentée à l'arrière de l'A4988 est trouée, alors le driver est grillé.



MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Code couleur résistances électriques