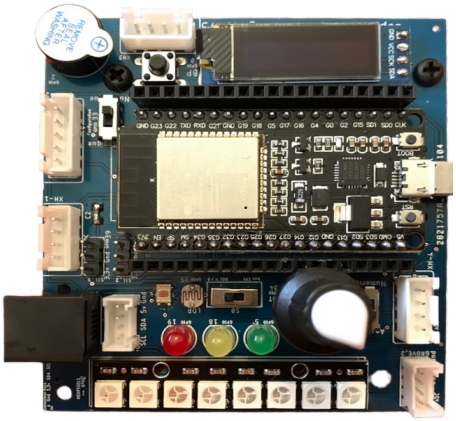
Utilisation des périphériques intégrés via les GPIO

***(General Purpose Input / Output)*.**

GPIO 19, 18, 5

Utilisation des **leds verte, jaune et rouge** :



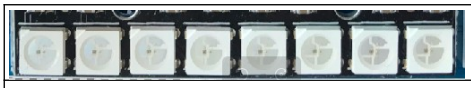
```
1  from machine import Pin
2  from time import sleep
3
4  led_v = Pin( 5, Pin.OUT ) # led verte connectée au GPIO 5
5  led_j = Pin( 18, Pin.OUT ) # led jaune
6  led_r = Pin( 19, Pin.OUT ) # led rouge
7
8  led_V.on( ) # allumer la led verte ( → configurer le GPIO à l'état haut )
9  sleep( 2 )
10  led.off( )  # éteindre la led verte ( → configurer le GPIO à l'état bas )
```

GPIO 0

Utilisation du **bouton poussoir** en mode 'simple' :

***Note :** Si le bouton est relâché, la valeur obtenue sera 1, s'il est enfoncé la valeur obtenue sera 0.*

```
1  from machine import Pin
2
3  bp = Pin( 0, Pin.IN ) # bouton poussoir connecté au GPIO 0
4
5  etat = bp.value( )  # lire l'état du bouton poussoir et affecter la valeur renvoyée à la variable etat
6
7  if etat == True :
8        print("Le bouton est relâché.")
9  else :
10       print("Le bouton est enfoncé.")
```



GPIO 12

Utilisation du **ruban de 8 leds NeoPixel**

```
1  from machine import Pin
2  from neopixel import NeoPixel
3  from time import sleep
4
5  ruban = NeoPixel( Pin(12), 8) # déclarer la variable ruban pour interagir avec les led NeoPixel
6  ruban.timing = (390, 800, 820, 430) # adaptation des temporisations → synchronisation des signaux
7
8  ruban[3] = ( 35, 35, 125) # Configurer la led d'indice 3 (la 4eme led du ruban) avec la couleur (35,35,125)
9  ruban.write( ) # envoyer l'information au ruban
10
11  sleep( 2 ) # attendre 2 s
12
13  for i in range(8) : # i prend pour valeurs successives :  0, 1, 2, 3, 4, 5, 6, 7
14     ruban[i] = (i*15, 105-i*15, 0) # configurer la couleur de chaque led : [ (0, 105, 0) , (15, 90, 0), ... ]
15  ruban.write( ) # envoyer la configuration au ruban
16
17  sleep( 1 )
18
19  for i in range(8) :
20     ruban[i] = (0, 0, 0) # aucune lumière → éteindre le ruban
21  ruban.write( )
```


GPIO 34 / GPIO 35 Utilisation du mode Conversion Analogique → Numérique (CAN)

Note : *Ce mode est utilisable pour le potentiomètre ou pour la LDR (Light Dependant Résistor)*

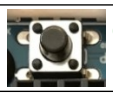
```
from machine import Pin ,ADC  # ADC → Analog to Digital Conversion ( CAN en français )

sensor = ADC ( Pin( 34 ) ) # configuration du GPIO 34 en mode ADC
sensor.atten ( ADC.ATTN_11DB ) # Atténuation de 11 dB → adaptation d'échelle si tension de 3,3V

val = sensor.read( )  # lire la valeur du CAN connecté au potentiomètre et l'affecter à la variable val
val_memo = val

print("Tournez le potentiomètre ...")
print("(Fin du script si la valeur obtenue est inférieure à 10) \n")  # \n → sauter une ligne
print("Valeur obtenue après mesure puis conversion : ", end="" ) # end="" → sur la même ligne
print("{:04d}".format(val), end="" ) # afficher la valeur sur 4 digits avec les zéro non significatifs

while val > 10 : # tant que la valeur obtenue est supérieure à 10
    val = sensor.read( ) # lire l'état du Conv. Analog. Numerique connecté au potentiomètre
    if val < val_memo-20 or val_memo+20 < val :
        val_memo = val
        print("\b"*4, end="") # effacer la précédente mesure (\b → backspace)
        print("{:04d}".format(val), end="" ) # mettre à jour l'affichage de la valeur
```



GPIO 0 Utilisation du **bouton poussoir** en mode **interruption matérielle** :

Plus d'infos : https://docs.micropython.org/en/latest/library/machine.Pin.html

```
""" L'utilisation des interruptions : irq (interrutpion request) permet d'éviter d'avoir à lire l'état
de la broche. La fonction (routine) de gestion de l'interruption va interagir avec le programme
principal (ici la routine affecte la valeur True à la variable globale bp_flag (drapeau)  )"""
from machine import Pin

def irq_flag ( pin ): # routine (fonction) de gestion de l'interruption
    global bp_flag # variable globale
    bp_flag = True # modification de la valeur affectée au drapeau

bp = Pin( 0 , Pin.IN )
bp.irq( trigger=Pin.IRQ_RISING, handler = irq_flag ) # utilisation de l'interruption sur front montant

bp_flag = False # variable globale (drapeau) aussi utilisée par la fonction de gestion d'interrutpion
while bp_flag == False :  # si appui sur le bouton : bp_flag <-True via la fonction irq_flag(...)
    pass  # ne rien faire
print("Ok pour interruption.")
```

GPIO 2 Utilisation du **buzzer** en mode PWM :

Plus d'infos : https://docs.micropython.org/en/latest/esp32/quickref.html#pwm-pulse-width-modulation

```
from machine import Pin, PWM
from time import sleep, sleep_ms # attente en secondes, sleep_ms → idem mais en millisecondes

buz = PWM ( Pin(2) ) # contrôle de la broche 2 en mode vibration : Pulse Wave Modulation

buz.freq( 440 ) # fréquence du signal : 440 Hz
sleep(1)
buz.duty(10) # diminuer l'intensité sonore à 10
sleep(1)
buz.duty(0) # intensité sonore à 0 → silence
sleep(1)

buz.deinit() # désactiver le buzzer
```