MicroPython pour ESP32

Bibliothèques et fonctions essentielles

Houssem-eddine LAHMER

15 mai 2025

Qu'est-ce que MicroPython ?

MicroPython

- Implémentation Python 3 optimisée pour les microcontrôleurs
- Développé par Damien George en 2013
- Langage interprété avec un sous-ensemble du langage Python standard
- REPL (Read-Eval-Print Loop) interactif
- Écosystème riche de bibliothèques et modules

ESP32 - Architecture et Caractéristiques

- SoC (System on Chip) double cœur 32-bits
- CPU : Xtensa LX6 à 240 MHz
- RAM : 520 KB SRAM
- Flash : 4-16 MB
- Wi-Fi 802.11 b/g/n
- Bluetooth 4.2 (BLE)
- 36 GPIO pins
- 18 canaux ADC 12-bits
- 2 DAC 8-bits
- 10 capteurs tactiles
- 4 SPI
- 2 I²S
- 2 I²C
- 3 UART
- CAN bus

Installation de MicroPython sur ESP32

1. Télécharger le firmware depuis `https://micropython.org/download/esp32/`
2. Effacer la flash (facultatif) :

```
esptool.py --chip esp32 --port /dev/ttyUSB0
    erase_flash

```

3. Flasher le firmware MicroPython :

```
esptool.py --chip esp32 --port /dev/ttyUSB0 --baud
    460800 \
write_flash -z 0x1000 esp32-XXXXXXXX.bin

```

Modules standards Python disponibles

Modules Python standard

- `array`
- `binascii`
- `collections`
- `errno`
- `gc`
- `hashlib`
- `heapq`
- `io`
- `json`
- `math`
- `os`
- `random`
- `re`
- `select`
- `socket`
- `ssl`
- `struct`
- `sys`
- `time`
- `zlib`

Module `machine`

machine

Module central pour accéder au hardware du microcontrôleur

- `Pin` - Contrôle des entrées/sorties GPIO
- `ADC` - Conversion analogique-numérique
- `DAC` - Conversion numérique-analogique
- `PWM` - Modulation de largeur d'impulsion
- `Timer` - Timers matériels
- `RTC` - Horloge temps réel
- `WDT` - Watchdog timer
- `TouchPad` - Interface tactile capacitive

E/S Numériques - Module machine.Pin (Sorties)

```
1  from machine import Pin
2
3  # Configurer un GPIO comme sortie
4  led = Pin(2, Pin.OUT)  # LED sur GPIO2
5
6  # Allumer/ teindre la LED
7  led.on()
8  led.off()
9  led.value(1)  # Allumer
10 led.value(0)  #  teindre
11
12 # Inverser l' tat
13 auto_state = not led.value()
14 led.value(auto_state)
15
```

E/S Numériques - Module machine.Pin (Entrées et IRQ)

```
1  from machine import Pin
2
3  # Configurer une entr e avec pull-up
4  button = Pin(0, Pin.IN, Pin.PULL_UP)
5
6  # Lire l' tat  du bouton
7  state = button.value()
8  print(" tat  du bouton :", state)
9
10 # Exemples pull-up / pull-down
11 pin_pu = Pin(4, Pin.IN, Pin.PULL_UP)
12 pin_pd = Pin(5, Pin.IN, Pin.PULL_DOWN)
13
14 # Configurer une interruption sur front descendant
15 def callback(pin):
16     print("Interruption d tect e sur pin", pin.pin)
17
18 button.irq(trigger=Pin.IRQ_FALLING,
19            handler=callback)
```

Convertisseur Analogique-Numérique (ADC) - Configuration

```
1  from machine import ADC, Pin
2
3  # ESP32: ADC disponible sur plusieurs pins
4  # R solution: 12 bits (0-4095)
5  adc = ADC(Pin(36))  # ADC1_CH0
6
7  # Configuration (ESP32)
8  adc.width(ADC.WIDTH_12BIT)  # R solution (9-12 bits)
9  adc.atten(ADC.ATTN_11DB)    # Att nuation (0, 2.5, 6,
       11dB)
10                             # 11dB permet mesure 0-3.3
       V
11
```

Convertisseur Analogique-Numérique (ADC) - Lecture

```
# Lecture de la valeur brute et conversion en tension
raw_value = adc.read()        # 0-4095
voltage = raw_value * 3.3 / 4095  # Conversion en
    volts

# ESP32: Lire temp rature interne du chip
from machine import ADC

temp_sensor = ADC(ADC.TEMP)
raw_temp = temp_sensor.read()
print("Temp. brute capteur interne :", raw_temp)

```

Convertisseur Numérique-Analogique (DAC) - Initialisation

Note ESP32

Sur ESP32, le DAC est disponible uniquement sur les pins 25 (DAC1) et 26 (DAC2).

```
1 from machine import DAC, Pin
2
3 # Cr er un objet DAC sur GPIO25 (DAC1)
4 dac = DAC(Pin(25))
5
6 #  crire  une valeur (0-255)
7 dac.write(128)  # ~1.65V (50% de 3.3V)
8
```

Convertisseur Numérique-Analogique (DAC) - Génération de formes d'onde

```
1  import math
2  import time
3
4  # G n rer une onde sinuso dale
5  def sine_wave():
6      for i in range(100):
7          # Sinus entre 0 et 255
8          value = int((math.sin(i/10 * math.pi) + 1) *
       127.5)
9          dac.write(value)
10         time.sleep_ms(10)
11
```

Convertisseur Numérique-Analogique (DAC) - Initialisation

Note ESP32

Sur ESP32, le DAC est disponible uniquement sur les pins 25 (DAC1) et 26 (DAC2).

```
1 from machine import DAC, Pin
2
3 # Cr er un objet DAC sur GPIO25 (DAC1)
4 dac = DAC(Pin(25))
5
6 #  crire  une valeur (0-255)
7 dac.write(128)  # ~1.65V (50% de 3.3V)
8
```

Convertisseur Numérique-Analogique (DAC) - Génération de formes d'onde

```
1  import math
2  import time
3
4  # G n rer une onde sinuso dale
5  def sine_wave():
6      for i in range(100):
7          value = int((math.sin(i/10 * math.pi) + 1) *
       127.5)
8          dac.write(value)
9          time.sleep_ms(10)
10
```

PWM (Pulse Width Modulation) - Configuration

```
1  from machine import Pin, PWM
2  import time
3
4  # Cr er un objet PWM sur
       GPIO2
5  pwm = PWM(Pin(2))
6
7  # Configurer la fr quence (
       Hz)
8  pwm.freq(1000)
9
10 # Configurer le rapport
       cyclique (0-1023 sur ESP32
       )
11 pwm.duty(512)   # 50%
12
13 # Arr ter le PWM
14 pwm.deinit()
15
```

Applications :

- Contrôle de luminosité LED
- Commande de moteurs
- Génération de signaux audio
- Contrôle de servomoteurs

PWM (Pulse Width Modulation) - Variation cyclique

```
1  # Varier l'intensit  d'une LED
2  from machine import PWM, Pin
3  import time
4
5  pwm = PWM(Pin(2))
6  pwm.freq(1000)
7
8  for i in range(0, 1024, 10):
9      pwm.duty(i)
10     time.sleep_ms(50)
11
12 # Nettoyage
13 pwm.deinit()
14
```

Notes techniques :

- Résolution : 10 bits (0-1023)
- Fréquence : 1 Hz à 40 MHz
- 16 canaux PWM disponibles sur ESP32

Timers

```
1  from machine import Timer
2
3  # Cr er un timer
4  timer = Timer(0)  # Timer 0
5
6  # Fonction de callback
7  def timer_callback(timer):
8      print("Timer d clench !")
9
10 # Mode p riodique (ms)
11 timer.init(period=1000, mode=Timer.PERIODIC, callback=
       timer_callback)
12
13 # Mode one-shot
14 timer.init(period=5000, mode=Timer.ONE_SHOT, callback=
       timer_callback)
15
16 # Arr ter le timer
17 timer.deinit()
```

RTC (Real Time Clock) – Configuration

```
1 from machine import RTC
2
3 # Cr er un objet RTC
4 rtc = RTC()
5
6 # R gler la date et l'heure
7 # (ann e , mois, jour, jour_semaine, heures, minutes,
    secondes, sec )
8 rtc.datetime((2025, 5, 15, 3, 14, 30, 0, 0))
9
```

RTC (Real Time Clock) – Lecture Synchronisation

```
1  # Lire la date et l'heure actuelle
2  date_time = rtc.datetime()
3  print(f"Date: {date_time[2]}/{date_time[1]}/{date_time
       [0]}")
4  print(f"Heure: {date_time[4]}:{date_time[5]}:{
       date_time[6]}")
5
6  # Exemple: synchronisation via NTP
7  import ntptime
8  try:
9      ntptime.settime()  # Synchronise avec pool.ntp.org
10     print("RTC synchronis e via NTP")
11 except:
12     print("Erreur de synchronisation NTP")
13
```

WDT (Watchdog Timer)

Watchdog

Redémarre automatiquement le système en cas de blocage du programme.

```
1  from machine import WDT
2  import time
3
4  # Initialiser le watchdog (timeout en ms)
5  wdt = WDT(timeout=5000)  # 5 secondes
6
7  # Dans la boucle principale
8  while True:
9      # Faire quelque chose...
10     print("Programme en cours d'ex cution...")
11
12     # Nourrir le watchdog pour  viter  le red marrage
13     wdt.feed()
```

Deep Sleep (Économie d'énergie)

```
1  import machine
2  import time
3
4  # Fonction pour entrer en deep sleep
5  def deep_sleep(sleep_time_ms):
6      print(f"Entr e en deep sleep pour {sleep_time_ms
       /1000} secondes")
7      time.sleep(1)  # Temps pour voir le message
8      machine.deepsleep(sleep_time_ms)
9
10 # V rifier la cause du r veil
11 wake_reason = machine.reset_cause()
12 print(f"Cause du r veil: {wake_reason}")
13
14 # Configurer un r veil avec timer
15 deep_sleep(10000)  # Deep sleep pour 10 secondes
16
17 # R veil par pin externe (exemple: bouton sur GPIO0)
18 wake_pin = machine.Pin(0, machine.Pin.IN, machine.Pin.
       PULL_UP)
```

TouchPad (Capteurs tactiles)

```
1  from machine import TouchPad, Pin
2  import time
3
4  # ESP32: capteurs tactiles sur pins sp cifiques
5  # TOUCH0: GPIO4    TOUCH5: GPIO12
6  # TOUCH1: GPIO0    TOUCH6: GPIO14
7  # TOUCH2: GPIO2    TOUCH7: GPIO27
8  # TOUCH3: GPIO15   TOUCH8: GPIO33
9  # TOUCH4: GPIO13   TOUCH9: GPIO32
10
11 # Cr er un objet TouchPad
12 touch = TouchPad(Pin(4))  # TOUCH0 sur GPIO4
13
14 # Lire la valeur (plus la valeur est basse, plus la
       pression est forte)
15 while True:
16     val = touch.read()
17     print(f"Valeur capteur: {val}")
18
19     # D tecter un toucher
```

I²C — Configuration et scan

```
1  from machine import Pin, I2C
2
3  # Cr ation du bus I2C (ESP32 supporte deux bus I2C)
4  # FREQ: fr quence d'horloge (typiquement 100kHz ou
       400kHz)
5  i2c = I2C(0,                          # Bus I2C #0
6            scl=Pin(22),                # GPIO22 pour SCL
7            sda=Pin(21),                # GPIO21 pour SDA
8            freq=400000)                # 400kHz
9
10 # Scanner les p riph riques
11 devices = i2c.scan()
12 print("P riph riques trouv s:", [hex(d) for d in
       devices])
13
```

I²C — Lecture et écriture

```
1  #  crire  sur un p riph rique
2  i2c.writeto(0x3C, b'\x00\x10\x20')
3
4  # Lire depuis un p riph rique
5  data = i2c.readfrom(0x3C, 5)  # Lire 5 octets
6
7  #  criture -lecture combin e
8  i2c.writeto_mem(0x3C, 0x10, b'\x00\x01')  #  crire
       l'adresse m moire 0x10
9  data = i2c.readfrom_mem(0x3C, 0x10, 2)    # Lire 2
     octets depuis 0x10
10
```

SPI — Configuration et sélection

```
1  from machine import Pin, SPI
2
3  # Configuration SPI sur ESP32
4  # HSPI: sck=14, mosi=13, miso=12
5  # VSPI: sck=18, mosi=23, miso=19
6  spi = SPI(1,                          # HSPI (1) ou VSPI
       (2)
7            baudrate=5000000,           # 5 MHz
8            polarity=0,                 # CPOL=0
9            phase=0,                    # CPHA=0
10           bits=8,                     # 8 bits
11           firstbit=SPI.MSB,           # Bit de poids fort
       en premier
12           sck=Pin(18),                # Horloge
13           mosi=Pin(23),               # Master Out Slave In
14           miso=Pin(19))               # Master In Slave Out
15
16 # S lection de p riph rique (Chip Select)
17 cs = Pin(5, Pin.OUT)
18 cs.value(1)  # D sactiver le p riph rique
```

SPI — Transmission de données

```
# Activer le CS et écrire des données
cs.value(0)
spi.write(b'\x12\x34\x56')              # Écrire 3 octets

# Lire des données
data = spi.read(3)                      # Lire 3 octets
data = spi.read(3, 0xFF)                # Lire en envoyant 0xFF comme dummy

# Écriture et lecture simultanées
result_buf = bytearray(2)
spi.write_readinto(b'\xAB\xCD', result_buf)
cs.value(1)  # Désactiver le périphérique

```

UART — Configuration et envoi

```
1  from machine import UART, Pin
2
3  # ESP32 poss de 3 UART (0, 1, 2)
4  # UART0 est g n ralement utilis  pour le REPL
5  uart = UART(2,                      # UART2
6              baudrate=115200,        # Vitesse (bps)
7              tx=Pin(17),             # TX sur GPIO17
8              rx=Pin(16),             # RX sur GPIO16
9              timeout=1000)           # Timeout en ms
10
11 # Envoyer des donn es
12 uart.write('Hello\n')
13 uart.write(b'Hello as bytes\n')
14
```

UART — Lecture et traitement

```
# V rifier si des donn es sont disponibles
if uart.any():
    # Lire des donn es
    data = uart.read()              # Lire tout ce qui est
    disponible
    data = uart.read(10)            # Lire 10 octets max
    line = uart.readline()          # Lire jusqu'   '\n'

    # Traiter les donn es re ues
    if data:
        print('Re u :', data)

```

OneWire et DS18x20 (Capteurs de température)

```
1  from machine import Pin
2  import onewire, ds18x20, time
3
4  # Cr er un bus OneWire
5  ow_pin = Pin(4)
6  ow_bus = onewire.OneWire(ow_pin)
7
8  # Cr er un objet DS18x20
9  ds_sensor = ds18x20.DS18X20(ow_bus)
10
11 # Scanner les capteurs sur le bus
12 sensors = ds_sensor.scan()
13 print("Capteurs trouv s:", [hex(int.from_bytes(s, '
       little')) for s in sensors])
14
15 # Lecture de temp rature
16 def read_temp():
17     ds_sensor.convert_temp()           # D marrer la
       conversion
18     time.sleep_ms(750)                 # Attendre la
```

BLE (Bluetooth Low Energy)

```
import bluetooth
from ble_simple_peripheral import BLESimplePeripheral
import time

# Cr er un objet BLE
ble = bluetooth.BLE()
ble_device = BLESimplePeripheral(ble)

# Callback pour r ception de donn es
def on_rx(data):
    print("Donn es re ues:", data.decode())

    # R pondre au client
    if data == b'get_temp':
        ble_device.send("Temperature: 25.5 C")

# Boucle principale
while True:
    if ble_device.is_connected():
        # Envoyer des donn es p riodiquement
```

WiFi - Configuration et connexion

```
import network
import time

# Configurer le WiFi en mode Station (client)
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)

# Scanner les r seaux disponibles
networks = sta_if.scan()
for net in networks:
    ssid, bssid, channel, rssi, authmode, hidden = net
    print(f"SSID: {ssid.decode()}, Signal: {rssi}dB")

# Se connecter    un r seau
sta_if.connect('SSID', 'PASSWORD')

# Attendre la connexion
while not sta_if.isconnected():
    print("Connexion en cours...")
    time.sleep(1)
```

WiFi - Mode Point d'Accès

```
1  import network
2  import time
3
4  # Configurer le WiFi en mode Access Point
5  ap_if = network.WLAN(network.AP_IF)
6  ap_if.active(True)
7
8  # Configurer le point d'acc s
9  ap_if.config(essid='ESP32-AP',              # Nom du
       r seau
10             authmode=network.AUTH_WPA2_PSK, #
       S curit
11             password='micropython',         # Mot de
       passe (min 8 caract res)
12             channel=1,                      # Canal WiFi
13             hidden=False)                   # R seau
       visible
14
15 # V rifier l' tat  du point d'acc s
16 print("Point d'acc s activ :", ap_if.active())
```

Requêtes HTTP avec urequests

```
1  import urequests
2  import json
3
4  # GET Request
5  response = urequests.get('http://api.example.com/data'
       )
6  print("Status:", response.status_code)
7
8  # Traiter la r ponse
9  if response.status_code == 200:
10     data = response.json()  # D code JSON
11     print("Donn es:", data)
12
13     # Ou r cup rer le texte brut
14     text = response.text
15     print("Texte:", text)
16
17     # Ou le contenu binaire
18     content = response.content
```

WebSocket Client

```
import uwebsocket
import network
import time

# D'abord se connecter au WiFi
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)
sta_if.connect('SSID', 'PASSWORD')
while not sta_if.isconnected():
    time.sleep(1)

# Cr er une connexion WebSocket
ws = uwebsocket.connect('ws://echo.websocket.org')

# Envoyer un message
ws.send('Hello WebSocket')

# Recevoir un message (bloquant)
response = ws.recv()
print("R ponse:", response)
```

Serveur Web Simple

```
1  import socket
2  import network
3  import machine
4  import time
5
6  # Configurer le WiFi (AP ou STA)
7  ap = network.WLAN(network.AP_IF)
8  ap.active(True)
9  ap.config(essid='ESP32-Web', password='micropython',
       authmode=3)
10
11 # Cr er un socket TCP
12 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13 s.bind(('', 80))
14 s.listen(5)
15
16 # LED pour montrer les requ tes
17 led = machine.Pin(2, machine.Pin.OUT)
18
19 # HTML de la page principale
```

MQTT (Message Queuing Telemetry Transport)

```
1  from umqtt.simple import MQTTClient
2  import machine
3  import time
4  import ubinascii
5
6  # Identifiant unique bas  sur l'ID du chip
7  client_id = ubinascii.hexlify(machine.unique_id())
8
9  # Configuration MQTT
10 mqtt_server = "broker.hivemq.com"
11 mqtt_port = 1883
12 mqtt_user = ""  # Si authentification requise
13 mqtt_password = ""  # Si authentification requise
14
15 # Topics
16 topic_sub = b"esp32/commands"
17 topic_pub = b"esp32/sensors"
18
19 # Callback pour messages re us
20 def sub_cb(topic, msg):
```

Resources et documentation

- Documentation officielle MicroPython : `https://docs.micropython.org/`
- Documentation spécifique ESP32 : `https://docs.micropython.org/en/latest/esp32/quickref.html`
- GitHub MicroPython : `https://github.com/micropython/micropython`
- Forums MicroPython : `https://forum.micropython.org/`
- Awesome MicroPython : `https://github.com/mcauser/awesome-micropython`
- MicroPython Cookbook par Packt Publishing

Bibliothèques additionnelles

Les bibliothèques suivantes peuvent être installées avec `upip` (MicroPython Package Manager) :

- `micropython-umqtt.simple` - Client MQTT
- `micropython-uasyncio` - Programmation asynchrone
- `micropython-urequests` - Requêtes HTTP