MicroPython sur ESP8266 ou ESP32

<markdown>

Premiers pas avec MicroPython sur l'ESP32

[Reference Micropython](https://docs.micropython.org/en/latest/esp32/quickref.html)

L'utilisation de MicroPython est un excellent moyen de tirer le meilleur parti de votre carte ESP32. vice versa, la puce ESP32 est une excellente plate-forme pour utiliser MicroPython. Le didacticiel vous guidera dans la configuration de MicroPython, l'obtention d'une invite et l'utilisation WebREPL, connexion au réseau et communication avec Internet, en utilisant les périphériques matériels et le contrôle de certains composants externes.

C'est parti !

Exigences

La première chose dont vous avez besoin est une carte avec une puce ESP32. Le MicroPython Le logiciel prend en charge la puce ESP32 elle-même et n'importe quelle carte devrait fonctionner. la caractéristique d'une carte est la façon dont les broches GPIO sont connectées à l'extérieur monde, et s'il comprend un convertisseur USB-série intégré pour rendre le UART disponible sur votre PC.

Les noms des broches seront donnés dans ce tutoriel en utilisant les noms des puces (par exemple GPIO2) et il devrait être simple de trouver à quelle broche cela correspond sur votre conseil particulier.

Alimentation de la carte

Si votre carte est équipée d'un connecteur USB, il est fort probable qu'elle soit alimentée par ceci lorsqu'il est connecté à votre PC. Sinon, vous devrez l'alimenter directement. Veuillez vous référer à la documentation de votre carte pour plus de détails.

Obtenir le firmware

La première chose à faire est de télécharger [le firmware](https://micropython.org/download/ESP32_GENERIC) MicroPython le plus récent .bin à charger sur votre appareil ESP32. Vous pouvez le télécharger à partir du [Page de téléchargement de MicroPython](https://micropython.org/download/ESP32_GENERIC/) À partir de là, vous avez 3 choix principaux :

* Versions de firmware stables * Versions quotidiennes du firmware * Versions quotidiennes du firmware avec prise en charge de SPIRAM

Si vous débutez avec MicroPython, le mieux est d'opter pour la version Stable versions de firmware. Si vous êtes un utilisateur avancé et expérimenté de MicroPython ESP32 qui aimerait suivre de près le développement et aider à tester de nouveaux fonctionnalités, il y a des builds quotidiennes. Si votre carte prend en charge SPIRAM, vous pouvez utilisez soit le firmware standard, soit le firmware avec support SPIRAM, et dans dans ce dernier cas, vous aurez accès à plus de RAM pour les objets Python.

Déploiement du firmware

Une fois que vous avez le firmware MicroPython, vous devez le charger sur votre appareil ESP32. Il y a deux étapes principales pour ce faire : vous devez d'abord mettre votre appareil en mode bootloader, et deuxièmement, vous devez copier le firmware. Le mode exact la procédure pour ces étapes dépend fortement de la carte en question et vous il faut se référer à sa documentation pour plus de détails.

Heureusement, la plupart des cartes disposent d'un connecteur USB, d'un convertisseur USB-série et du DTR et les broches RTS câblées d'une manière spéciale, le déploiement du firmware devrait être aussi simple que toutes les étapes peuvent être effectuées automatiquement. Les cartes qui ont de telles fonctionnalités incluent les Adafruit Feather HUZZAH32, M5Stack, Wemos LOLIN32 et TinyPICO cartes, ainsi que les kits de développement Espressif DevKitC, PICO-KIT, WROVER-KIT.

Pour de meilleurs résultats, il est recommandé d'effacer d'abord l'intégralité du flash de votre appareil avant d'installer le nouveau firmware MicroPython.

Actuellement, nous ne prenons en charge que la copie d'esptool.py dans le firmware. Vous pouvez trouver cet outil [ici](https://github.com/espressif/esptool/)

en utilisant pip::

```
  pip installe esptool
```

Les versions commençant par 1.3 prennent en charge Python 2.7 et Python 3.4 (ou plus récent). Une version plus ancienne (au moins 1.2.1 est nécessaire) fonctionne bien mais nécessitera Python 2.7.

En utilisant esptool.py, vous pouvez effacer le flash avec la commande :

```
  esptool.py --port /dev/ttyUSB0 effacer_flash
  ou sous windows esptool.py --port COM3 effacer_flash
```

Et puis déployez le nouveau firmware en utilisant ::

```
  esptool.py --chip esp32 --port /dev/ttyUSB0 write_flash -z 0x1000
esp32-20180511-v1.9.4.bin
```

Remarques :

* Vous devrez peut-être modifier le paramètre « port » en quelque chose d'autre adapté à votre

```
PC
```

* Vous devrez peut-être réduire le débit en bauds si vous obtenez des erreurs lors du flashage

```
(par exemple jusqu'à 115200 en ajoutant « --baud 115200 » dans la commande)
```

* Pour certaines cartes avec une configuration FlashROM particulière, vous devrez peut-être

```
changer le mode flash (par exemple en ajoutant ``-fm dio`` dans la commande)
```

* Le nom de fichier du firmware doit correspondre au fichier que vous avez

Si les commandes ci-dessus s'exécutent sans erreur, MicroPython doit être installé sur votre planche !

Invite série

Une fois que vous avez le firmware sur l'appareil, vous pouvez accéder au REPL (invite Python) via UART0 (GPIO1=TX, GPIO3=RX), qui peut être connecté à un port USB-série convertisseur, selon votre carte. Le débit en bauds est de 115200.

À partir de là, vous pouvez maintenant suivre le tutoriel ESP8266, car ces deux puces Espressif sont très similaires lorsqu'il s'agit d'utiliser MicroPython sur eux. Le tutoriel ESP8266 se trouve dans :ref:`esp8266_tutorial` (mais ignorez la section Introduction).

Dépannage des problèmes d'installation

Si vous rencontrez des problèmes lors du flashage ou de l'exécution immédiate du firmware après cela, voici des recommandations de dépannage :

* Soyez attentif et essayez d'exclure les problèmes matériels. Il existe 2 problèmes courants

```
Problèmes : mauvaise qualité de la source d'alimentation et FlashROM
usée/défectueuse.
En parlant de source d'alimentation, non seulement l'ampérage brut est
important, mais aussi son faible
ondulation et bruit/EMI en général. L'alimentation la plus fiable et la plus
pratique
la source est un port USB.
```

* Les instructions de clignotement ci-dessus utilisent une vitesse de clignotement de 460800 bauds, ce qui est

```
bon compromis entre vitesse et stabilité. Cependant, selon votre
module/carte, convertisseur USB-UART, câbles, système d'exploitation hôte,
etc., le débit ci-dessus
Le débit peut être trop élevé et entraîner des erreurs. Essayez un débit
plus courant de 115 200 bauds.
Dans de tels cas, il est préférable d'utiliser un tarif plus élevé.
```

* Pour détecter un contenu flash incorrect (par exemple provenant d'un secteur défectueux sur une puce),

```
ajoutez le commutateur « --verify » aux commandes ci-dessus.
```

* Si vous rencontrez toujours des problèmes lors du flashage du firmware, veuillez

```
se référer à [la page du projet
esptool.py](https://github.com/espressif/esptool)
pour une documentation supplémentaire et un outil de suivi des bogues où
vous pouvez signaler les problèmes.
```

* Si vous parvenez à flasher le firmware mais que l'option ``–verify`` renvoie

```
des erreurs même après plusieurs tentatives peuvent survenir, vous avez
peut-être une puce FlashROM défectueuse.
```

Modulation de largeur d'impulsion

La modulation de largeur d'impulsion (PWM) est un moyen d'obtenir une sortie analogique artificielle sur un broche numérique. Cela se fait en basculant rapidement la broche de bas en haut. Il y a deux paramètres associés à cela : la fréquence du basculement, et le cycle de service. Le cycle de service est défini comme la durée pendant laquelle la broche est haute par rapport à la durée d'une seule période (temps bas et temps haut). Maximum le cycle de service est lorsque la broche est haute tout le temps, et le minimum est lorsqu'elle est bas tout le temps.

* Exemple plus complet avec les 16 canaux PWM et les 8 temporisateurs ::

```
  à partir de l'importation de la machine Pin, PWM
  essayer:
      f = 100 # Hz
      d = 1024 // 16 # 6,25%
      broches = (15, 2, 4, 16, 18, 19, 22, 23, 25, 26, 27, 14, 12, 13, 32,
33)
      pwms = []
      pour i, épingle dans enumerate(pins) :
          pwms.append(PWM(Pin(pin), freq=f * (i // 2 + 1), duty= 1023 si
i==15 sinon d * (i + 1)))
          imprimer(pwms[i])
  enfin:
      pour pwm dans pwms :
          essayer:
              pwm.deinit()
          sauf:
              passer
```

```
La sortie est :
```

```
    * PWM (Pin (15), fréquence = 100, service = 64, résolution = 10, mode =
0, canal = 0, minuterie = 0)
*
*     PWM (Pin (2), fréquence = 100, service = 128, résolution = 10, mode =
0, canal = 1, minuterie = 0)
*
*     PWM (Pin (4), fréquence = 200, service = 192, résolution = 10, mode =
0, canal = 2, minuterie = 1)
*
*     PWM (broche (16), fréquence = 200, service = 256, résolution = 10,
mode = 0, canal = 3, minuterie = 1)
*
*     PWM (Pin (18), fréquence = 300, service = 320, résolution = 10, mode =
0, canal = 4, minuterie = 2)
*
*     PWM (broche (19), fréquence = 300, service = 384, résolution = 10,
mode = 0, canal = 5, minuterie = 2)
*
*     PWM (Pin (22), fréquence = 400, service = 448, résolution = 10, mode =
0, canal = 6, minuterie = 3)
*
*     PWM (Pin (23), fréquence = 400, service = 512, résolution = 10, mode =
0, canal = 7, minuterie = 3)
*
*     PWM (Pin (25), fréquence = 500, service = 576, résolution = 10, mode =
1, canal = 0, minuterie = 0)
*
*     PWM (Pin (26), fréquence = 500, service = 640, résolution = 10, mode =
1, canal = 1, minuterie = 0)
*
*     PWM (broche (27), fréquence = 600, service = 704, résolution = 10,
mode = 1, canal = 2, minuterie = 1)
*
*     PWM (broche (14), fréquence = 600, service = 768, résolution = 10,
mode = 1, canal = 3, minuterie = 1)
*
*     PWM (broche (12), fréquence = 700, service = 832, résolution = 10,
mode = 1, canal = 4, minuterie = 2)
*
*     PWM (broche (13), fréquence = 700, service = 896, résolution = 10,
mode = 1, canal = 5, minuterie = 2)
*
*     PWM (Pin (32), fréquence = 800, service = 960, résolution = 10, mode =
1, canal = 6, minuterie = 3)
*
*     PWM (Pin (33), fréquence = 800, service = 1023, résolution = 10, mode
= 1, canal = 7, minuterie = 3)
```

* Exemple de changement de fréquence en douceur ::

from time import sleep from machine import Pin, PWM

DUTY_MAX = 2**16 - 1

duty_u16 = 0 delta_d = 16

p = PWM(Pin(5), 1000, duty_u16=duty_u16) print(p)

while True:

```
  p.duty_u16(duty_u16)
```

```
  sleep(1 / 1000)
```

```
  duty_u16 += delta_d
  if duty_u16 >= DUTY_MAX:
      duty_u16 = DUTY_MAX
      delta_d = -delta_d
  elif duty_u16 <= 0:
      duty_u16 = 0
      delta_d = -delta_d
```

```
Voir l'onde PWM sur la broche (5) avec un oscilloscope.
```

Remarque : il n'est pas nécessaire de spécifier le mode Pin.OUT. Le canal est initialisé en mode PWM en interne une fois pour chaque broche transmise au constructeur PWM.

Le code suivant est erroné ::

```
  pwm = PWM(Pin(5, Pin.OUT), freq=1000, duty=512) # Pin(5) en mode PWM ici
  pwm = PWM(Pin(5, Pin.OUT), freq=500, duty=256) # Pin(5) en mode OUT ici,
PWM est désactivé
```

Utilisez plutôt ce code ::

```
  pwm = PWM(Pin(5), freq=1000, duty=512)
  pwm.init(fréquence=500, service=256)
```

</markdow n>

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

Permanent link:
https://chanterie37.fr/fablab37110/doku.php?id=debuter_en_python:micropython2&rev=1740067869

Last update: **2025/02/20 17:11**