MicroPython pour Raspberry Pi PICO avec prise en charge Ethernet

Table of Contents

Raspberry Pi Pico n'a pas de réseau par défaut, mais un support USB Ethernet et Ethernet PHY existe. Le support **PHY** a été construit autour de la pile **lwIP**[1)], il exploite les capacités PIO, DMA et double cœur du RP2040 pour créer une pile MAC Ethernet dans le logiciel:

- les modules comme le **Microchip LAN8720** prennent actuellement en charge PHY Ethernet basés sur RMII.
- le convertisseur Ethernet vers UART **Pico-ETH-CH9121**, permet la communication réseau Ethernet 10/100M via UART
- les cartes WIZnet Ethernet HAT et WIZnet W5100S-EVB-Pico

Construction à partir des sources

Pour compiler micropython, le PC doit utiliser un environnement Linux ou Unix avec :

- CMake (plus que la version 3.12)
- Thonny (qui facilite l'utilisation de micropython)

Si la chaîne d'outils Raspberry Pi Pico est déjà configurée et fonctionne, il faut s'assurer que **pico-sdk** est à jour, y compris les sous-modules. Si ce n'est pas le cas, il faut d'abord configurer le SDK C/C++.

Installer Thonny IDE sur Raspberry Pi Pico: `sudo apt install thonny`

Carte de dérivation LAN8720 basée sur RMII

Le mappage entre le numéro de broche physique RP2040 avec la carte de dérivation LAN8720 basée sur RMII est le suivant:

Pico	**RP20401**	**LAN8720 Breakout**
Broche 9	GP6	RX0

Pico	RP20401	LAN8720 Breakout
Broche 10	GP7	RX1 (RX0 + 1)
Broche 11	GP8	CRS (RX0 + 2)
Broche 14	GP10	TX0
Broche 15	GP11	TX1 (TX0 + 1)
Broche 16	GP12	TX-EN (TX0 + 2)
Broche 19	GP14	MDIO
Broche 20	GP15	MDC
Broche 26	GP20	nINT / RETCLK
3V3 (SORTIE)	—	VCC
Broche 38	GND	GND

Ces broches sont la définies dans bibliothèque par défaut et peuvent être modifiées dans le logiciel.

Récupérer le projet pico-rmii-ethernet sur GitHub, ainsi que la pile lwIP.

```
git clone git@github.com:sandeepmistry/pico-rmii-ethernet.git
cd pico-rmii-ethernet
git submodule update --init
```

Il faut que **PICO_SDK_PATH** soit défini avant de continuer. Par exemple, si on construit sur un Raspberry Pi et qu'on a exécuté le script **pico_setup.sh**, il faut faire pointer le **PICO_SDK_PATH** vers :

```
export PICO_SDK_PATH=/home/pi/pico/pico-sdk
```

On peut continuer et créer à la fois la bibliothèque et l'exemple d'application.

```
mkdir build
cd build
cmake ..
make
```

Si tout se passe bien, on doit avoir un fichier UF2 dans `build/examples/httpd` appelé `pico_rmii_ethernet_httpd.uf2`. On peut maintenant charger ce fichier UF2 sur le Pico de la manière habituelle.

Brancher le câble sur le Raspberry Pi hôte, puis appuyer sur le bouton BOOTSEL du Pico et le maintenir enfoncé pendant que l'on branche l'autre extrémité du câble micro USB sur la carte. Relâcher ensuite le bouton une fois la carte branchée.

Un volume de disque appelé RPI-RP2 devrait apparaître sur le bureau. Double-cliquer pour l'ouvrir, puis faire glisser et déposer le fichier UF2. Le Pico exécute maintenant un serveur Web.

Carte Pico-ETH-CH9121

Le mappage entre le numéro de broche physique RP2040 avec la carte Pico-ETH-CH9121 est le suivant:

I/O	Pico	RP20401	Pico-ETH-CH9121
	Broche 39	—	Entrée d'alimentation 5V VSYS
		GND	GND Masse
I	Broche 0	GP0	RXD1 UART 1 entrée de données
O	Broche 2	GP1	TXD1 UART 1 Sortie de données
I	Broche 6	GP4	RXD2 UART 2 Entrée de données
O	Broche 7	GP5	TXD2 UART 2 Sortie de données
	Broche 19	GP14	CFG0 broche d'activation de configuration réseau
	Broche 22	GP17	RST1 Broche de réinitialisation

Les modes UDP sont les mêmes que TCP, la seule différence est que le mode est UDP CLIENT/SERVER mais pas TCP CLIENT/SERVER.

Récupérer les sources du projet :

```
sudo apt-get install p7zip-full
cd ~
sudo wget https://www.waveshare.com/w/upload/a/a4/Pico_ETH_CH9121_CODE.7z
7z x Pico_ETH_CH9121_CODE.7z -o./Pico_ETH_CH9121_CODE
cd ~/Pico_ETH_CH9121_CODE
cd Pico/c/build/
```

Il faut que **PICO_SDK_PATH** soit défini avant de continuer. Par exemple, si on construit sur un Raspberry Pi et qu'on a exécuté le script **pico_setup.sh**, il faut faire pointer le **PICO_SDK_PATH** vers :

```
export PICO_SDK_PATH=/home/pi/pico/pico-sdk
```

continuer et créer à la fois la bibliothèque et l'exemple d'application.

```
cmake ..
make -j9
```

Après la construction, copier le fichier .uf2 sur le Pico

Appuyer sur le bouton BOOTSEL de Pico et le maintenir enfoncé, connecter le pico au Pi par câble micro USB, puis relâcher le bouton. Lorsque le disque portable RPI-RP2 est reconnu, copier le fichier uf2 sur le disque portable.

```
cp main.uf2 /media/pi/RPI-RP2/
```

Brancher le Pico sur Ethernet et également via USB sur l'hôte. En plus d'alimenter le Pico, on peut voir des informations de débogage via USB Serial. Ouvrir une fenêtre de terminal et démarrer minicom.

```
minicom -D /dev/ttyACM0
```

Si le routeur distribue des adresses IP, on devrait voir quelque chose comme ceci dans la fenêtre minicom, montrant que le Pico a récupéré une adresse IP en utilisant DHCP :

```
pico rmii ethernet - httpd
netif status changed 0.0.0.0
netif link status changed up
netif status changed 192.168.1.110
```

Si on ouvre une fenêtre de navigateur et tape l'adresse IP que le routeur a attribuée au Pico dans la barre d'adresse, si tout se passe bien, on devrait voir la page d'index lwIP par défaut.

Pour configurer le module, il suffit de modifier le Serial Port Parameter Configuration.py:

```
ODE = 1 #0:TCP Server 1:TCP Client 2:UDP Server 3:UDP Client
GATEWAY = (169, 254, 88, 1) # GATEWAY
TARGET_IP = (169, 254, 88, 17)# TARGET_IP
LOCAL_IP = (169,254,88,70) # LOCAL_IP
SUBNET_MASK = (255,255,255,0) # SUBNET_MASK
LOCAL_PORT1 = 5000 # LOCAL_PORT1
LOCAL_PORT2 = 4000 # LOCAL_PORT2
TARGET_PORT = 3000 # TARGET_PORT
BAUD_RATE = 115200 # BAUD_RATE
```

Cartes WIZnet

Ethernet-HAT a la même disposition et le même espacement des broches que le Raspberry Pi Pico. Les W5100S et RJ45 sont intégrés, Ethernet peut être utilisé en se branchant sur le Raspberry pi pico. Une chose à noter lors de l'utilisation de HAT est de regarder attentivement la direction et de la brancher. Il y a une forme USB marquée, et cette direction et la direction USB de Pico doivent être les mêmes.

Dans la carte W5100S-EVB-Pico, les broches GPIO sont connectées de la même manière que la carte Raspberry Pi Pico. Si Pico utilise Ethernet, les broches PIO16, GPIO17, GPIO18, GPIO19, GPIO20 et GPIO21 ne peuvent pas être utilisées. C'est une broche utilisée à l'intérieur de la carte RP2040.

I/O	Pico	W5100S
I	GPIO16	MISO
O	GPIO17	CSn

I/O	Pico	W5100S
O	GPIO18	SCLK
O	GPIO19	MOSI
O	GPIO20	RSTn
I	GPIO21	INTn
I	GPIO24	Détection VBUS - Up si VBUS est présent, sinon Down
O	GPIO25	Connecté à l'utilisateur LED
I	GPIO29	Utilisé en mode ADC (ADC3) pour mesurer VSYS/3

On peut trover le firmware précompilé sur
https://github.com/Wiznet/RP2040-HAT-MicroPython/releases

Cloner les bibliothèques Ethernet avec la commande Git suivante.

```
make -rf RP2040
cd /RP2040
git clone https://github.com/Wiznet/RP2040-HAT-MicroPython.git
```

Appliquer manuellement les correctifs téléchargés avec la commande Git dans chaque répertoire de bibliothèque.

- 0001-Added-WIZnet-Chip-library.patch : Ethernet (puce WIZnet)
- 0002-Added-AXTLSlibrary.patch : SSL/TLS (AXTLS)

```
cd /RP2040/RP2040-HAT-MicroPython
cmake CMakeLists.txt
```

Editer le ficheir Makefile pour vérifier que le patch c'est bien déroulé:

```
  cd libraries/ports/rp2
  vi Makefile
```

si un échec se produit pendant le patch ,on ne devrait pas voir le code ci-dessous dans le Makefile.

```
  MICROPY_PY_WIZNET5K ?= 5105
  //and
  CMAKE_ARGS += -DMICROPY_PY_WIZNET5K=$(MICROPY_PY_WIZNET5K)
```

Pour le w5100s et rp2040, utiliser le code ci-dessous.

```
make
```

Pour le w5500 et le rp2040, utiliser le code ci-dessous.

```
make MICROPY_PY_WIZNET5K=5500
```

Brancher le Raspberry Pi Pico en maintenant le bouton BOOTSEL enfoncé.

Le lecteur s'appellera RPI-RP2 sur toutes les cartes RP2040. Télécharger et placer le fichier UF2 (firmware.uf2) dans Pico.

On peut également utiliser **thonny**, cliquer sur MicroPython (Raspberry Pi Pico) dans la barre d'état et choisit “Configurer l'interpréteur...” pour accéder au menu d'installation du firmware.

Les paramètres de l'interprète s'ouvriront. Cliquer sur Installer ou mettre à jour le firmware.

brancher le Raspberry Pi Pico tout en maintenant le bouton BOOTSEL enfoncé à l'invite.

Ensuite, cliquer sur Installer. Attendre que l'installation soit terminée et cliquer sur Fermer.

Programmation

Carte de dérivation LAN8720 basée sur RMII

Il est facile de changer les pages Web servies par Pico. On peut trouver le “système de fichiers” avec les pages lwIP par défaut dans l'application HTTP dans le sous-module lwIP Git.

```
cd pico-rmii-ethernet/lib/lwip/src/apps/http/fs
ls
404.html   img/        index.html
```

Il faut modifier le fichier `index.html` in situ ici avec un éditeur préféré. Ensuite, il faut déplacer le répertoire du système de fichiers en place, puis on peut le reconditionner à l'aide du script **makefsdata** associé.

```
cd ..
mv fs makefsdata
cd makefsdata
perl makefsdata
```

L'exécution de ce script créera un fichier `fsdata.c` dans le répertoire courant. Il faut déplacer ce fichier vers le répertoire parent, puis reconstruire le fichier UF2.

```
mv fsdata.c ..
```

```
cd ../../../../../..
rm -rf build
mkdir build
cd build
cmake ..
make
```

Si tout se passe bien, on devrait avoir un nouveau fichier UF2 dans `build/examples/httpd` appelé `pico_rmii_ethernet_httpd.uf2` que l'on peut à nouveau chargersur le Pico comme avant.

Au redémarrage, attendre que le Pico récupère à nouveau une adresse IP, puis, en ouvrir à nouveau une fenêtre de navigateur et en taper l'adresse IP attribuée à votre Pico dans la barre d'adresse, on doit maintenant voir une page Web mise à jour.

On peut revenir en arrière et modifier la page servie à partir du Pico et créer un site entier. Il faut reconstruire le fichier `fsdata.c` à chaque fois avant de reconstruire votre UF2.

imites actuelles: Il y a quelques limites à l'implémentation actuelle. Le RP2040 fonctionne sous-cadencé à seulement 50 MHz en utilisant l'horloge de référence des modules RMII, tandis que la pile lwIP est compilée avec **NO_SYS** afin que ni l'API Netcon ni l'API Socket ne soient activées. Enfin, la vitesse de liaison est définie sur 10 Mbps car il existe actuellement un problème avec TX à 100 Mbps.

Carte Pico-ETH-CH9121

Cette section présente quelques paramètres qui peuvent être utilisées dans les codes C.

- Types de données:

```
#define UCHAR caractère non signé
#define UBYTE uint8_t
#define UWORD uint16_t
#define UDOUBLE uint32_t
```

- Initialisation du module :

```
void CH9121_init(void)
```

- Paramètres de configuration des modules

```
UCHAR CH9121_Mode //Mode
UCHAR CH9121_LOCAL_IP[4] //IP de l'appareil
UCHAR CH9121_GATEWAY[4] //Passerelle
UCHAR CH9121_SUBNET_MASK[4] //Masque de sous-réseau
UCHAR CH9121_TARGET_IP[4] //IP cible
UWORD CH9121_PORT1 //Port de l'appareil
```

```
UWORD CH9121_TARGET_PORT //Port cible
UDOUBLE CH9121_BAUD_RATE // débit en bauds de la série
```

- fonctions pour configurer le module avec des commandes série

```
void CH9121_TX_4_bytes(UCHAR data, int command);  //Peut être utilisé pour
configurer le mode, le port aléatoire, déconnecter le réseau, effacer le
tampon, DHCP, UART2
void CH9121_TX_5_bytes(UWORD data, int command);//Peut être utilisé pour
définir le port de série
void CH9121_TX_7_bytes(UCHAR data[], int command);//Peut être utilisé pour
définir l'IP, le masque de sous-réseau, la passerelle.
void CH9121_TX_BAUD (données UDOUBLE, int command); // Peut être utilisé
pour définir le débit en bauds de Serial.
void CH9121_Eed() ); // Peut être utilisé pour enregistrer le réglage dans
l'EEPROM, activer le réglage, réinitialiser le module, quitter le mode de
réglage
```

Cartes WIZnet

Des exemples Ethernet sont disponibles dans le répertoire 'RP2040-HAT-MicroPython/examples/'. Pour certains d'entre eux, des bibliothèques de fonction doivent être ajoutées:

- **umqttsimple** un simple client MQTT (protocole de messagerie publish-subscribe basé sur le protocole TCP/IP) pour **MicroPython**: https://github.com/micropython/micropython-lib/blob/master/micropython/umqtt.simple/umqtt/simple.py
- **urequest** un module python permettant d'utiliser le protocole http de façon ultra simple: https://github.com/micropython/micropython-lib/blob/master/python-ecosys/urequests/urequests.py

thonny permet également d'importer des nouvelles bibliothèques de fonction.

Créer un nouveau fichier, sur l'ordinateur hôte avec le nom que l'on veut, par exemple “(votre nom de bibliothèque).py”
Aller à **Ouvrir> Cet ordinateur**
Le fichier doit être enregistré sur RPi Pico avec le nom “(votre nom de bibliothèque).py”
Aller dans **Fichier >Enregistrer sous> Raspberry Pi Pico**

test Ethernet

w5x00_Ping_Test.py permet de vérifier que le réseau est connecté normalement et les données sont envoyées les unes aux autres.

définir l'adresse IP dans le même sous réseau utilisé pour connecté l'hôte

Copier le contenu dans code.py sur votre RPi Pico et exécuter.

Un test de ping permet de voir les paquets sont échangés entre les hôtes.

Si on regarde le temps et le taux de perte parmi les résultats statistiques, on peut connaître l'état du réseau Internet.

Loopback

Pour tester l'exemple **Loopback**, des réglages mineurs doivent être effectués dans le code `Loopback/W5x00_Loopback.py`

Configurer SPI et réinitialiser la broche.

```
spi=SPI(0,2_000_000, mosi=Pin(19),miso=Pin(16),sck=Pin(18))
```

définir l'adresse IP dans le même sous réseau utilisé pour connecté l'hôte

```
nic = network.WIZNET5K(spi,Pin(17),Pin(20)) #spi,cs,reset pin
nic.ifconfig(('192.168.1.20','255.255.255.0','192.168.1.1','8.8.8.8'))
while not nic.isconnected():
    time.sleep(1)
    print(nic.regs())
```

Pour faire fonctionner la loopback en tant que serveur.

```
def server_loop():
    s = socket()
    s.bind(('192.168.1.20', 5000)) #Source IP Address
    s.listen(5)
    conn, addr = s.accept()
    print("Connect to:", conn, "address:", addr)
    print("Loopback server Open!")
    while True:
        data = conn.recv(2048)
```

```
        print(data.decode('utf-8'))
        if data != 'NULL':
            conn.send(data)
```

Pour faire fonctionner la loopback en tant que client:

```
def client_loop():
    s = socket()
    s.connect(('192.168.1.11', 5000)) #Destination IP Address
    print("Loopback client Connect!")
    while True:
        data = s.recv(2048)
        print(data.decode('utf-8'))
        if data != 'NULL' :
            s.send(data)
```

Télécharger le frichier et exécuter:

- Mode serveur de bouclage
  - Le bouclage est exécuté et le serveur attend dans l'état d'écoute.
  - Ouvrir le programme Hercules pour définir [Adresse IP] et [Numéro de PORT] et pour se connecter au serveur.
- Mode client de bouclage
  - Ouvrer le serveur dans le programme Hercules. Entrer le numéro de port et appuyer sur Écouter, le serveur s'ouvrira.
  - Lorsque la fonction Client est extraite de la source de bouclage et exécutée, elle se connecte normalement au serveur.
  - On peut voir qu'il est connecté depuis le serveur Hercules [Client Connect status].

Si on envoie des données, on peut voir qu'elles sont envoyées et reçues.

DHCP

Pour tester l'exemple DHCP, des réglages mineurs doivent être effectués dans le code `DHCP/dhcp.py`

définir l'adresse IP dans le même sous réseau utilisé pour connecté l'hôte

```
import network
from machine import Pin,SPI

spi=SPI(0,2_000_000, mosi=Pin(19),miso=Pin(16),sck=Pin(18))
```

```
nic = network.WIZNET5K(spi,Pin(17),Pin(20))

# If you use the Dynamic IP(DHCP), you must use the "nic.active(True)".
# If you use the Static IP, you must use the
"nic.ifconfig("IP","subnet","Gateway","DNS")".
    #
nic.ifconfig(('192.168.100.13','255.255.255.0','192.168.100.1','8.8.8.8'))
nic.active(True)
```

UPIP

```
import network
import usocket
from machine import Pin,SPI
import upip
import time

spi=SPI(0,2_000_000, mosi=Pin(19),miso=Pin(16),sck=Pin(18))
nic = network.WIZNET5K(spi,Pin(17),Pin(20))
nic.active(True)
time.sleep(0.1)
upip.install("micropython-urequests")
```

HTTP

La bibliothèque **urequests** doit être ajoutée, créer un nouveau fichier en appuyant sur le bouton Nouveau fichier. Copier le code de la bibliothèque **urequests** récupéré dans le lien suivant : https://github.com/micropython/micropython-lib/blob/master/python-ecosys/urequests/urequests.py

Pour tester l'exemple **Webserver**, des réglages mineurs doivent être effectués dans le code `HTTP/HTTP_Server/HTTP_Server.py`

Configurer SPI et réinitialiser la broche.

```
spi=SPI(0,2_000_000, mosi=Pin(19),miso=Pin(16),sck=Pin(18))
```

définir l'adresse IP dans le même sous réseau utilisé pour connecté l'hôte

```
nic = network.WIZNET5K(spi,Pin(17),Pin(20)) #spi,cs,reset pin
```

```
nic.ifconfig(('192.168.1.20','255.255.255.0','192.168.1.1','8.8.8.8'))
while not nic.isconnected():
    time.sleep(1)
    print(nic.regs())
```

Requête HTML

```
html = """
    <!DOCTYPE html>
    <html lang="en">
    <head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Raspberry Pi Pico Web server - WIZnet W5100S</title>
    </head>
    <body>
    <div align="center">
    <H1>Raspberry Pi Pico Web server & WIZnet Ethernet HAT</H1>
    <h2>Control LED</h2>
    PICO LED state: <strong>""" + led_state + """</strong>
    <p><a href="/?led=on"><button class="button">ON</button></a><br>
    </p>
    <p><a href="/?led=off"><button class="button
button2">OFF</button></a><br>
    </p>
    </div>
    </body>
    </html>
    """
```

```
Exécutez Pico pour ouvrir le serveur Web.
```

```
while True:
        conn, addr = s.accept()
        print('Connect from %s' % str(addr))
        request = conn.recv(1024)
        request = str(request)
        #print('Content = %s' % request)
        led_on = request.find('/?led=on')
        led_off = request.find('/?led=off')
        if led_on == 6:
            print("LED ON")
            led.value(1)
        if led_off == 6:
            print("LED OFF")
            led.value(0)
        response = web_page()
        conn.send('HTTP/1.1 200 OK\n')
```

```
        conn.send('Connection: close\n\n')
        conn.send(response)
        conn.close()
```

Télécharger et exécuter.

Si on ouvre la page Web HTML et entre 192.168.1.20 et on verra le serveur connecté

Pour tester l'exemple **Webclient**, des réglages mineurs doivent être effectués dans le code `HTTP/HTTP_Client/HTTP_Client.py`:

Configurer SPI et réinitialiser la broche.

```
spi=SPI(0,2_000_000, mosi=Pin(19),miso=Pin(16),sck=Pin(18))
```

définir l'adresse IP dans le même sous réseau utilisé pour connecté l'hôte

```
nic = network.WIZNET5K(spi,Pin(17),Pin(20)) #spi,cs,reset pin
nic.ifconfig(('192.168.1.20','255.255.255.0','192.168.1.1','8.8.8.8'))
while not nic.isconnected():
    time.sleep(1)
    print(nic.regs())
```

Requête HTML.

```
def request():
    #get
    r = urequest.get("http://httpbin.org/get")
    r.raise_for_status()
    print(r.status_code)
    print(r.text)
    #post
    r = urequest.post("http://httpbin.org/post", json={'Hello': 'WIZnet'})
    print(r.json())
```

Télécharger et exécuter

Accéder à l'adresse du serveur. Après cela, il accède au serveur dans chaque URL et imprime le contenu.

MQTT

La bibliothèque **umqttsimple** doit être ajoutée, créer un nouveau fichier en appuyant sur le bouton Nouveau fichier. Copier le code de la bibliothèque **umqttsimple** récupéré dans le lien suivant : https://github.com/micropython/micropython-lib/blob/master/micropython/umqtt.simple/umqtt/simple.py

Pour tester l'exemple **publish**, des réglages mineurs doivent être effectués dans le code `MQTT/Publish/MQTT_pub.py`

Configurer SPI et réinitialiser la broche.

```
spi=SPI(0,2_000_000, mosi=Pin(19),miso=Pin(16),sck=Pin(18))
```

définir l'adresse IP dans le même sous réseau utilisé pour connecté l'hôte

```
nic = network.WIZNET5K(spi,Pin(17),Pin(20)) #spi,cs,reset pin
nic.ifconfig(('192.168.1.20','255.255.255.0','192.168.1.1','8.8.8.8'))
while not nic.isconnected():
    time.sleep(1)
    print(nic.regs())
```

Dans la configuration MQTT, l'adresse IP du courtier est l'adresse IP du bureau.

```
#mqtt config
mqtt_server = '192.168.1.11'
client_id = 'wiz'
topic_pub = b'hello'
topic_msg = b'Hello Pico'

last_message = 0
message_interval = 5
counter = 0

def mqtt_connect():
    client = MQTTClient(client_id, mqtt_server, keepalive=60)
    client.connect()
    print('Connected to %s MQTT Broker'%(mqtt_server))
    return client
```

utiliser MQTT Publish.

```
#MQTT Publish
def main():
    w5x00_init()
    try:
        client = mqtt_connect()
    except OSError as e:
        reconnect()
    while True:
        client.publish(topic_pub, topic_msg)
        time.sleep(3)
    client.disconnect()
```

Télécharger et exécuter

Créer un courtier à l'aide de moustique en exécutant la commande suivante. Si le courtier est créé normalement, l'adresse IP du courtier est l'adresse IP actuelle de l'ordinateur hôte, et le port est 1883 par défaut.

```
mosquitto -c mosquitto.conf -p 1883 -v
```

Si l'exemple de publication MQTT fonctionne normalement sur Raspberry Pi Pico, on doit voir les informations réseau de Raspberry Pi Pico, se connecter au courtier et publier le message.

Pour tester l'exemple **publish**, des réglages mineurs doivent être effectués dans le code `MQTT/Publish/MQTT_pub.py`

Configurer SPI et réinitialiser la broche.

```
spi=SPI(0,2_000_000, mosi=Pin(19),miso=Pin(16),sck=Pin(18))
```

définir l'adresse IP dans le même sous réseau utilisé pour connecté l'hôte

```
nic = network.WIZNET5K(spi,Pin(17),Pin(20)) #spi,cs,reset pin
nic.ifconfig(('192.168.1.20','255.255.255.0','192.168.1.1','8.8.8.8'))
while not nic.isconnected():
    time.sleep(1)
    print(nic.regs())
```

Dans la configuration MQTT, l'adresse IP du courtier est l'adresse IP de l'hôte.

```
#mqtt config
mqtt_server = '192.168.1.11'
```

```
client_id = 'wiz'
topic_sub = b'hello'

last_message = 0
message_interval = 5
counter = 0

#MQTT connect
def mqtt_connect():
    client = MQTTClient(client_id, mqtt_server, keepalive=60)
    client.set_callback(sub_cb)
    client.connect()
    print('Connected to %s MQTT Broker'%(mqtt_server))
    return client
```

Pour utiliser MQTT subscribe.

```
#subscribe
def main():
    w5x00_init()
    try:
        client = mqtt_connect()
    except OSError as e:
        reconnect()

    while True:
        client.subscribe(topic_sub)
        time.sleep(1)

    client.disconnect()
```

Télécharger et exécuter

Créer un courtier à l'aide de moustique en exécutant la commande suivante. Si le courtier est créé normalement, l'adresse IP du courtier est l'adresse IP actuelle de l'hôte, et le port est 1883 par défaut.

```
mosquitto -c mosquitto.conf -p 1883 -v
```

Si l'exemple de publication MQTT fonctionne normalement sur Raspberry Pi Pico, vous pouvez voir les informations réseau de Raspberry Pi Pico, se connecter au courtier et publier le message.

Pour s'abonner au courtier utiliser la commande suivante: `moustique_sub -h 192.168.1.20 -t bonjour -m "Bonjour Pico"`

Dans les versions de **Mosquitto** antérieures à 2.0, la valeur par défaut est d'autoriser les clients à se connecter sans authentification. Dans les versions 2.0 et ultérieures, il faut choisir explicitement les options d'authentification avant que les clients puissent se connecter. Par conséquent, si on utilise la version 2.0 ou ultérieure, se il faut configurer 'mosquitto.conf' dans le répertoire où Mosquitto est installé.

[1)]

lwIP est une petite implémentation indépendante de la suite de protocoles **TCP/IP**. L'objectif de l'implémentation de **lwIP TCP/IP** est de réduire l'utilisation de la RAM tout en ayant un TCP à grande échelle. Cela rend **lwIP** adapté à une utilisation dans les systèmes embarqués avec des dizaines de kilo-octets de RAM libre et de la place pour environ 40 kilo-octets de code ROM.

From:
http://www.ouarte.garden/ - **dwndoc**

Permanent link:
http://www.ouarte.garden/doku.php?id=prive:pico-micropython-ethernet

Last update: **2025/02/19 10:59**