*Programmation avec µPython*

## Installation de µPython

On vérifie :

- la connexion de l'ESP32 ;
- la capacité de la mémoire flash ;

xterm

```
$ esptool.py chip_id
esptool.py v3.0
Found 2 serial ports
Serial port /dev/ttyUSB0
Connecting....
Detecting chip type... ESP32
Chip is ESP32-D0WDQ6 (revision 0)
Features: WiFi, BT, Dual Core, Coding Scheme None
Crystal is 26MHz
MAC: 24:0a:c4:83:30:60
Uploading stub...
Running stub...
Stub running...
Warning: ESP32 has no Chip ID. Reading MAC instead.
MAC: 24:0a:c4:83:30:60
Hard resetting via RTS pin...

$ esptool.py flash_id
esptool.py v3.0
Found 2 serial ports
Serial port /dev/ttyUSB0
Connecting........_
Detecting chip type... ESP32
Chip is ESP32-D0WDQ6 (revision 0)
Features: WiFi, BT, Dual Core, Coding Scheme None
Crystal is 26MHz
MAC: 24:0a:c4:83:30:60
Uploading stub...
Running stub...
Stub running...
Manufacturer: ef
Device: 4016
Detected flash size: 4MB
Hard resetting via RTS pin...
```

*vérification de l'ESP32*

*vérification du composant mémoire*

On efface le contenu de la mémoire flash :

xterm

```
$ esptool.py --chip esp32 --baud 460800 erase_flash
esptool.py v3.0
Found 2 serial ports
Serial port /dev/ttyUSB0
Connecting....
Chip is ESP32-D0WDQ6 (revision 0)
Features: WiFi, BT, Dual Core, Coding Scheme None
Crystal is 26MHz
MAC: 24:0a:c4:83:30:60
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Erasing flash (this may take a while)...
Chip erase completed successfully in 8.2s
Hard resetting via RTS pin...
```

*effacement de la mémoire de l'ESP32*

On flashe la mémoire avec le firmware $\mu$Python :

xterm

```
$ esptool.py --chip esp32 --baud 460800 write_flash -z 0x1000
esp32spiram-idf4-20210202-v1.14.bin
esptool.py v3.0
Found 2 serial ports
Serial port /dev/ttyUSB0
Connecting.....
Chip is ESP32-D0WDQ6 (revision 0)
Features: WiFi, BT, Dual Core, Coding Scheme None
Crystal is 26MHz
MAC: 24:0a:c4:83:30:60
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Compressed 1571824 bytes to 981125...
Wrote 1571824 bytes (981125 compressed) at 0x00001000 in 24.0 seconds (effective
524.9 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

*flashage du firmware $\mu$Python*

## Configuration du Raspberry Pi

Pour savoir si le WiFi fonctionne :

xterm

```
$ rfkill unblock wlan
$ sudo ip l set wlan0 up
$ sudo iw dev wlan0 scan | grep SSID:
```

Supprimer l'utilisation du WiFi en mode client :

xterm

```
sudo systemctl disable wpa_supplicant
```

Pour la mise en place du point d'accès WiFi :

xterm

```
$ sudo apt install dnsmasq
$ sudo systemctl disable dnsmasq
$ sudo apt install hostapd
$ sudo systemctl disable hostapd
$ sudo reboot
```

Le script `hotspot` à créer sur le Raspberry Pi :

```
#!/bin/bash
INTERFACEWAN=eth0
INTERFACE=wlan0
SSID=IoT
PSK=12344321

PREFIX=10.33.33

CONFIG=$(cat <<END
interface=$INTERFACE
hw_mode=g
macaddr_acl=0
auth_algs=3
# Disable this to insure the AP is visible:
ignore_broadcast_ssid=0
channel=6
ssid=$SSID
wpa=2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
wpa_passphrase=$PSK
END
)

ip l set dev $INTERFACE down
ip l set dev $INTERFACE up
ip a flush dev $INTERFACE
```

```
hostapd <(echo "$CONFIG") &
ip a add $PREFIX.254/24 dev $INTERFACE
sysctl net.ipv4.ip_forward=1
iptables -t nat -A POSTROUTING -s $PREFIX.0/24 -o $INTERFACEWAN  -j MASQUERADE
dnsmasq -d -z -a $PREFIX.254 -F $PREFIX.100,$PREFIX.150,255.255.255.0 -O 6,$PRE
FIX.254 -A /serveur.iot.com/$PREFIX.254 -l /tmp/leases
```

Pour l'exécuter :

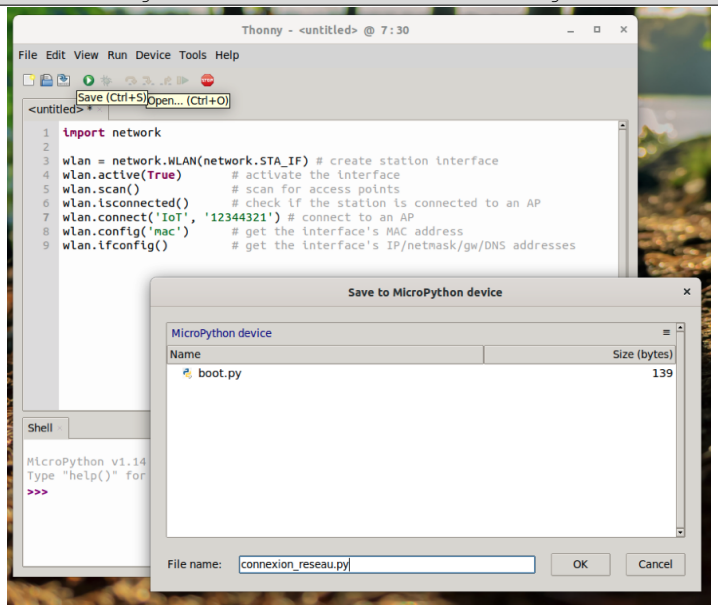
xterm

```
$ chmod +x hotspot
$ sudo ./hotspot
Configuration file: /dev/fd/63
wlan0: Could not connect to kernel driver
net.ipv4.ip_forward = 1
Using interface wlan0 with hwaddr b8:27:eb:eb:90:87 and ssid "IoT"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
dnsmasq: started, version 2.76 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt DBus i18n IDN DHCP DHCPv6 no-Lua
TFTP conntrack ipset auth DNSSEC loop-detect inotify
dnsmasq-dhcp: DHCP, IP range 10.33.33.100 -- 10.33.33.150, lease time 1h
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 8.8.8.8#53
dnsmasq: using nameserver 8.8.4.4#53
dnsmasq: read /etc/hosts - 5 addresses
```

Sur l'ESP32 :

```
import network

wlan = network.WLAN(network.STA_IF) # create station interface
wlan.active(True)       # activate the interface
wlan.scan()             # scan for access points
wlan.isconnected()      # check if the station is connected to an AP
wlan.connect('essid', 'password') # connect to an AP
wlan.config('mac')      # get the interface's MAC address
wlan.ifconfig()         # get the interface's IP/netmask/gw/DNS addresses
```




Ce qui produit sur le Raspberry Pi :

xterm

```
wlan0: STA 24:0a:c4:83:30:60 IEEE 802.11: associated
wlan0: AP-STA-CONNECTED 24:0a:c4:83:30:60
wlan0: STA 24:0a:c4:83:30:60 RADIUS: starting accounting session B5CAD8028304EFCB
wlan0: STA 24:0a:c4:83:30:60 WPA: pairwise key handshake completed (RSN)
dnsmasq-dhcp: DHCPDISCOVER(wlan0) 24:0a:c4:83:30:60
dnsmasq-dhcp: DHCPOFFER(wlan0) 10.33.33.140 24:0a:c4:83:30:60
dnsmasq-dhcp: DHCPDISCOVER(wlan0) 24:0a:c4:83:30:60
dnsmasq-dhcp: DHCPOFFER(wlan0) 10.33.33.140 24:0a:c4:83:30:60
dnsmasq-dhcp: DHCPDISCOVER(wlan0) 24:0a:c4:83:30:60
dnsmasq-dhcp: DHCPOFFER(wlan0) 10.33.33.140 24:0a:c4:83:30:60
dnsmasq-dhcp: DHCPREQUEST(wlan0) 10.33.33.140 24:0a:c4:83:30:60
dnsmasq-dhcp: DHCPACK(wlan0) 10.33.33.140 24:0a:c4:83:30:60 espressif
```

### Utilisation du serveur IoT en requête http

xterm
```
$ sudo apt install python3-pip
$ pip3 install gevent
$ pip3 install bottle
```

Vous installerez le programme du TD précédent :

xterm
```
$ git clone https://git.p-fb.net/pef/iot_bottle_sse.git
```

Vous déploierez le code du fichier `requete_iot_lumiere_esp32.py` sur l'ESP32.

Sur le Raspberry Pi :

xterm
```
$ python3 serveur_chart.py
Bottle v0.12.19 server starting up (using GeventServer())...
Listening on http://:8080/
Hit Ctrl-C to quit.
```

### MQTT : serveur sur le Raspberry Pi et client sur l'ESP32

xterm
```
$ sudo apt install mosquitto mosquitto-clients
```

Pour vérifier que le serveur Mosquitto est bien démarré :

xterm
```
$ ss -tln
State              Recv-Q     Send-Q     Local Address:Port         Peer Address:Port
LISTEN             0          32          10.33.33.254:53               0.0.0.0:*
LISTEN             0          128              0.0.0.0:22               0.0.0.0:*
LISTEN             0          100              0.0.0.0:1883             0.0.0.0:*
LISTEN             0          128                 [::]:22                  [::]:*
LISTEN             0          100                 [::]:1883 Le serveur MQTT [::]:*
```

Écrire un programme $\mu$Python utilisant le module `MQTTClient` transmettant la valeur du « *Hall sensor* » périodiquement.

```
from umqtt.simple import MQTTClient

client = MQTTClient("ESP32", mqtt_serveur)
topic = "HallSensor"
client.connect()
```

le topic

*Vous trouverez la commande pour publier dans la documentation du module MQTTClient.*

Sur le Raspberry Pi, on va pouvoir souscrire au topic et obtenir les valeurs du capteur :

xterm
```
$ mosquitto_sub -t 'HallSensor'
```