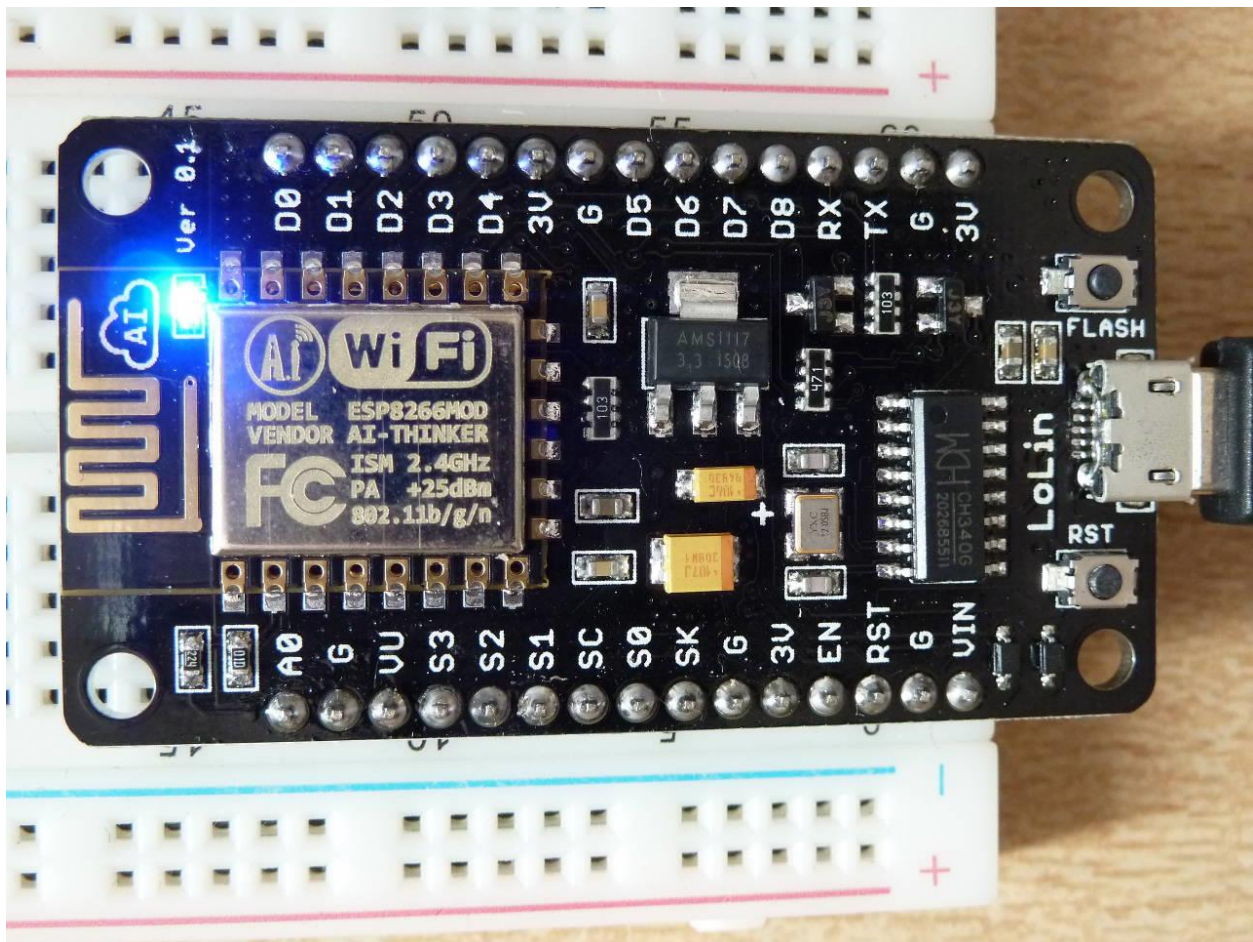


*Mise en route d'une carte WeMos-LoLin avec le firmware
NodeMCU et un module WiFi ESP8266*

Voir aussi l'article sur l'ESP8266 Amica



Firmware NodeMCU

http://nodemcu.com/index_en.html

Référence de la carte chez Banggood

<http://www.banggood.com/V3-NodeMcu-Lua-WIFI-Development-Board-p-992733.html?p=0431091025639201412F>

AliExpress propose des versions moins chères avec un chip `silabs cp2102` au lieu du `CH340G` pour la communication USB.

Cette carte utilise un ESP8266 ESP-12E (d'après l'apparence). Voir les différentes versions de l'ESP8266.

Référence fabricant



La carte est donc une *LoLin* du fabricant WeMos

La page de la carte en question : www.wemos.cc/wiki/

L'ESP8266 a été développé par Espressif. D'après cet article, il semblerait que l'ESP8266 a d'abord été vendu comme une passerelle UART pour les microcontrôleurs, mais qu'en fait il contient un processeur 32-bit Tensilica Xtensa LX106 à 80 MHz avec un "full WiFi stack".

Communication USB

⚠ Cette carte utilise une puce CH340G pour la communication USB. Pour l'installation du pilote sur OSX, voir l'article *Utiliser un chip CH340G au lieu d'un FTDI*.

Sur le site du fabricant, il y a deux drivers pour Mac :

Mac (old) ⇒ MD5 = 505487fe1033a9485f2e3fb0520718e8

Mac (new) ⇒ MD5 = f2c61b093909d6d54f6a466e7e367a39

La version "old" a le même MD5 que celle disponible à http://www.wch.cn/download/CH341SER_MAC_ZIP.html.

Il me semble que la version "new" ne nécessite pas d'enlever la protection d'OSX contre les pilotes non signés ⇒ à vérifier.

Adresse de la carte

Sur mon MacBook Pro, l'adresse de la carte est `/dev/tty.wchusbserial1410` sur le port USB de gauche ou `/dev/tty.wchusbserial1420` sur le port USB de droite.

À noter que je ne branche jamais de cartes de développement sur mon clavier externe, car ce clavier est un hub sans alimentation qui consomme donc du courant sur le port USB. Quand on connecte une carte, il y a le risque que le Mac fasse un reboot comme si on lui avait enlevé l'alimentation et comme ça m'est déjà arrivé, je ne recommence plus.

D'ailleurs, les ports USB sous-alimentés du Raspberry II ne supportent pas ce clavier pour la même raison : il consomme trop (50 mA mesuré).

```
ls /dev | grep tty.wchusbserial # retourne par ex : tty.wchusbserial1410
ls /dev/tty.wchusbserial1410 # adresse OK si cette commande retourne
quelque chose
```

Mise à jour du firmware

Installer `esptool`

```
git clone https://github.com/themadinventor/esptool.git
cd esptool
python setup.py install
which esptool.py # ⇒ /usr/local/bin/esptool.py
```

Télécharger le dernier firmware

Par défaut la carte est livrée avec le firmware floating point

La dernière version (nodemcu_float_0.9.6-dev_20150704.bin) fonctionne pour moi.

<http://www.wemos.cc/wiki/Tutorial/Downloads#firmware>

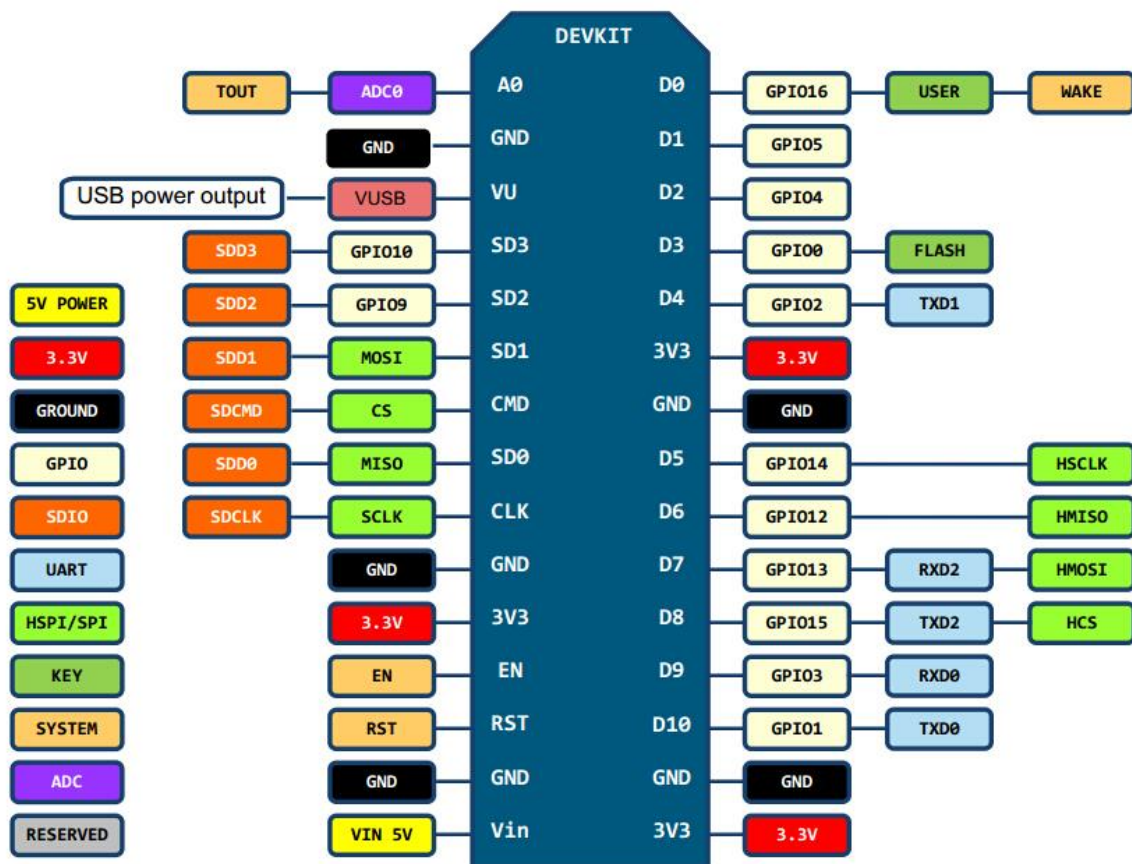
Flasher le firmware

```
USBPORT=/dev/tty.wchusbserial1410
FIRMWARE=nodemcu_float_0.9.6-dev_20150704.bin
esptool.py \
  --port $USBPORT \
  --baud 230400 \
  write_flash \
  --flash_mode qio \
  --flash_size 32m \
  --flash_freq 40m \
  0x000000 $FIRMWARE
```

Pinout

ESP8266 – LoLin

www.wemos.cc/wiki/



D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

Programmation de l'ESP8266

L'ESP8266 peut être programmé de plusieurs façons :

- En Arduino C (voir ci-dessous)
- En C, avec le SDK d'Espressif
- En C, avec Sming
- En Lua (voir ci-dessous)
- En JavaScript, avec le firmware Espruino
- En MicroPython
- En BASIC

Programmation de l'ESP8266 en Arduino C

Les informations d'origine se trouvent ici : <https://github.com/esp8266/Arduino/>

Configuration de l'IDE Arduino

À faire une fois

- Télécharger et installer, l'IDE Arduino
- Ouvrir les préférences de l'IDE Arduino
- Cliquer sur "URL de gestionnaire de cartes supplémentaires" et coller le lien suivant :

```
http://arduino.esp8266.com/stable/package\_esp8266com\_index.json
```

- Installer la bibliothèque "esp8266 by ESP8266 Community" (Outils/Type de carte/Gestionnaire de carte)

La bibliothèque est installée au chemin suivant :

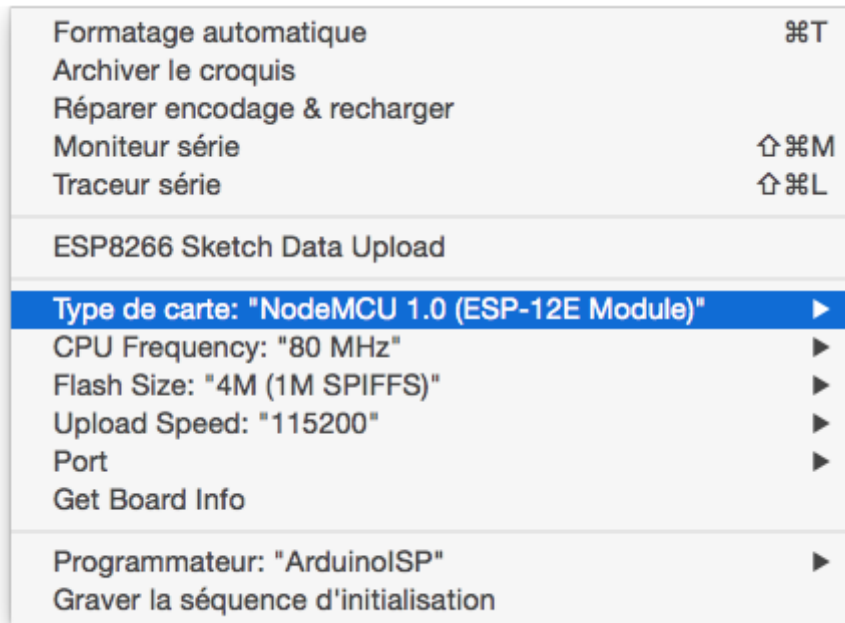
```
~/Library/Arduino15/packages/esp8266/hardware/esp8266/
```

Note : Adafruit et LoLin proposent d'autres gestionnaires. J'ai choisi la version de la communauté qui maintient le site esp8266.com. La version d'Adafruit est un fork de celle d'esp8266.com.

Configuration de la carte dans l'IDE

À faire pour chaque ESP différent. Exemple pour le LoLin (ESP-12E).

Dans le menu `Outils`



L'ESP8266 peut être overclocké à 160 MHz. J'ai observé que la librairie `Tone.h` n'aime pas ça...

Compilation et téléchargement des programmes

La compilation et le téléchargement se font exactement comme pour une carte Arduino. Pour la programmation, il peut y avoir quelques différences de fonctionnalité liées aux différences du hardware et à l'implémentation des bibliothèques. Il y a quelques informations sur le GitHub d'ESP8266 et Adafruit a aussi documenté quelques différences sur leur *fork*.

Gestionnaire de téléchargement *esp8266fs*

L'ESP8266 dispose d'un système de fichiers SPIFFS et on peut donc télécharger des fichiers depuis un ordinateur sur l'ESP8266 via l'IDE Arduino. Pour ce faire, il faut installer un gestionnaire de téléchargements appelé *esp8266fs*. Les instructions se trouvent ici :

<https://github.com/esp8266/arduino-esp8266fs-plugin/>

Il y a également quelques informations sur cette page :

<http://esp8266.github.io/Arduino/versions/2.0.0/doc/filesystem.html#uploading-files-to-file-system>

Ce gestionnaire de téléchargement ne sert pas à programmer l'ESP8266 (voir Compilation et téléchargement pour cette fonctionnalité), mais à télécharger des fichiers annexes, typiquement des fichiers qui seront servis par une interface web (HTML, images...) ou des fichiers de configuration.

L'utilisation est très simple :

- Créer un répertoire `data` dans le dossier du croquis.
- Y copier les fichiers.
- Utiliser la fonction `ESP8266 Sketch Data Upload` dans le menu `Outils` (voir image du menu `Outils` ci-dessus).

Le gros problème de cette façon de faire, c'est que le téléchargement prend beaucoup de temps quelle que soit la taille du répertoire `data`, car l'entier de la mémoire disponible est flashé.

À 115'200 bauds, il faut environ 2 min pour une *Flash Size SPIFFS* d'1 Mo et 6 min pour une *Flash Size SPIFFS* de 3 Mo (réglage `Flash Size` dans le menu `Outils`, voir image ci-dessus). On peut diminuer ces temps d'un facteur 2 en utilisant une vitesse de transmission de 230'400 bauds. Au delà l'IDE génère une erreur. À noter que même à 115'200 bauds, l'IDE plante de temps en temps et qu'il est nécessaire de le redémarrer et de débrancher et rebrancher l'ESP8266 également.

Exemples de programme

Blink ESP8266

Affiche quelques caractéristiques de l'ESP8266 dans la console série

Notes

- Sur OSX, les bibliothèques sont installées dans le répertoire `~/Library/Arduino15/packages/`.
- Quelques variables utiles sont définies dans le fichier `~/Library/Arduino15/packages/esp8266/hardware/esp8266/2.2.0/variants/nodemcu/pins_arduino.h`.

Programmation de l'ESP8266 en Lua

Les scripts Lua peuvent être interprétés ou compilés avec le firmware NodeMCU.

On trouve quantité de programmes sur le web pour charger les scripts Lua sur l'ESP8266. J'ai testé les suivants :

- explorer
- esp8266 (Node.js)

- luatool
- nodemcu-uploader
- upload à la main

Esplorer

Esplorer est un programme java avec une interface graphique assez moche. Au-delà de son aspect, il est rapide et stable et c'est celui que j'utilise en ce moment.

<http://esp8266.ru/esplorer/>

<http://esp8266.ru/esplorer-latest/?f=ESPlorer.zip>

esp8266 (Node.js)

Il faut avoir installé Node.js au préalable. Ne fonctionne pas avec la version 4.2.1 de Node.js, mais fonctionne avec la version 0.12.7.

<https://www.npmjs.com/package/esp8266>

La vitesse de transmission est fixée à 9600 bits/s et ne peut pas être changée.

```
# Installation
npm install esp8266 -g

# Définition du port
esp port set /dev/tty.wchusbserial1410

# Liste des fichiers
esp file list

# Chargement d'une version compressée des scripts
esp file push init.lua
```

luatool

<https://github.com/4refr0nt/luatool>

```
git clone https://github.com/4refr0nt/luatool.git
#!/bin/bash
```

```
# upload.sh ⇒ upload les scripts Lua avec luatool
# https://github.com/4refr0nt/luatool
```

```
USBPORT=/dev/tty.wchusbserial1420
```

```
./luatool/luatool/luatool.py \
  --port $USBPORT           \
  --src init_1.lua           \
  --dest init.lua            \
  --verbose                  \
  --restart
```

nodemcu-uploader

<https://github.com/kmpm/nodemcu-uploader>

```
git clone https://github.com/kmpm/nodemcu-uploader.git
#!/bin/bash
```

```
# upload.sh ⇒ upload les scripts Lua avec nodemcu-uploader
# https://github.com/kmpm/nodemcu-uploader
```

```
USBPORT=/dev/tty.wchusbserial1420
BAUD=115200
```

```
./nodemcu-uploader/nodemcu-uploader.py \
  --port $USBPORT \
  --baud $BAUD \
  upload \
  init1.lua:init.lua \
  --restart
```

À la main

On peut aussi envoyer le fichier ligne par ligne à la main avec des logiciels comme screen ou CoolTerm

Par défaut la vitesse de communication est de 9600 bits/s. Avec luatool, on ne peut aller qu'à cette vitesse. Avec nodemcu-uploader on peut aller plus haut. J'ai testé à 115200 bits/s et ça fonctionne, mais la communication se bloque de temps en temps. Il me semble qu'il est plus sage de travailler uniquement à 9600 bits/s.

Hello World

Créer un fichier `init_1.lua`

```
-- init_1.lua
-- Fait clignoter la LED de l'ESP8266
led1 = 4
gpio.mode( led1, gpio.OUTPUT )
for i=1,10 do
  gpio.write( led1, gpio.HIGH )
  tmr.delay( 0.1E6 )
  gpio.write( led1, gpio.LOW )
  tmr.delay( 1E6 )
end
print( "Coucou, c'est moi !" )
```

Pour l'upload, créer un fichier `upload.sh` (voir ci-dessus).

Alternativement, il est possible d'envoyer le fichier ligne par ligne avec screen ou CoolTerm.

```
file.open( "init.lua", "w" )
file.writeline([[ led1 = 4                                     ]])
file.writeline([[ gpio.mode( led1, gpio.OUTPUT )             ]])
file.writeline([[ for i=1,10 do                               ]])
file.writeline([[      gpio.write( led1, gpio.HIGH )          ]])
file.writeline([[      tmr.delay( 0.1E6 )                     ]])
file.writeline([[      gpio.write( led1, gpio.LOW )           ]])
file.writeline([[      tmr.delay( 1E6 )                       ]])
file.writeline([[ end                                         ]])
file.writeline([[ print( "Coucou, c'est moi !" )              ]])
file.close()
```

Exemple avec un mini serveur web

<http://randomnerdtutorials.com/esp8266-web-server/>

IOT

- dweet.io
- dweet + NodeMCU
- Freeboard
- <https://www.openhomeautomation.net/internet-of-things-dashboard/>

Liens

- NodeMCU API sur GitHub
- NodeMCU API sur nodemcu.com
- <https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout?view=all>
- <http://neilkolban.com/tech/esp8266/>
- <http://randomnerdtutorials.com/home-automation-using-esp8266/>
- <https://www.hackster.io/rayburne/esp8266-01-using-arduino-ide>
- <http://www.esp8266.com>
- <http://frightanic.com>

- Comparison of ESP8266 NodeMCU development boards
- eLua project
- <http://blog.nyl.io/esp8266-motor/>
- <http://hackaday.com/2015/03/18/how-to-directly-program-an-inexpensive-esp8266-wifi-module/>
- <http://benlo.com/esp8266/esp8266QuickStart.html>
- Building NodeMCU from Source for the ESP8266
- Introduction to the MQTT Protocol on NodeMCU
- ESP8266 WiFi File Management
- Commandes AT
- 4 reasons I abandoned NodeMCU/Lua for ESP8266, (24th April 2015)
- 4 ways to eliminate ESP8266 resets

Nico