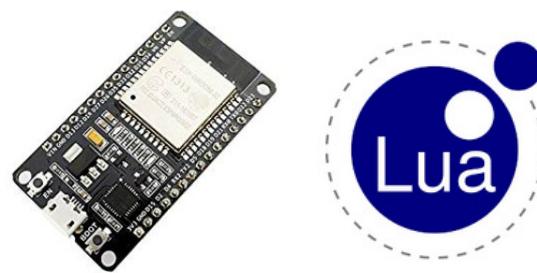


IOT : *Internet Of Things*

Introduction aux microcontrôleurs

Gildas Ménier
Maître de conférences en informatique



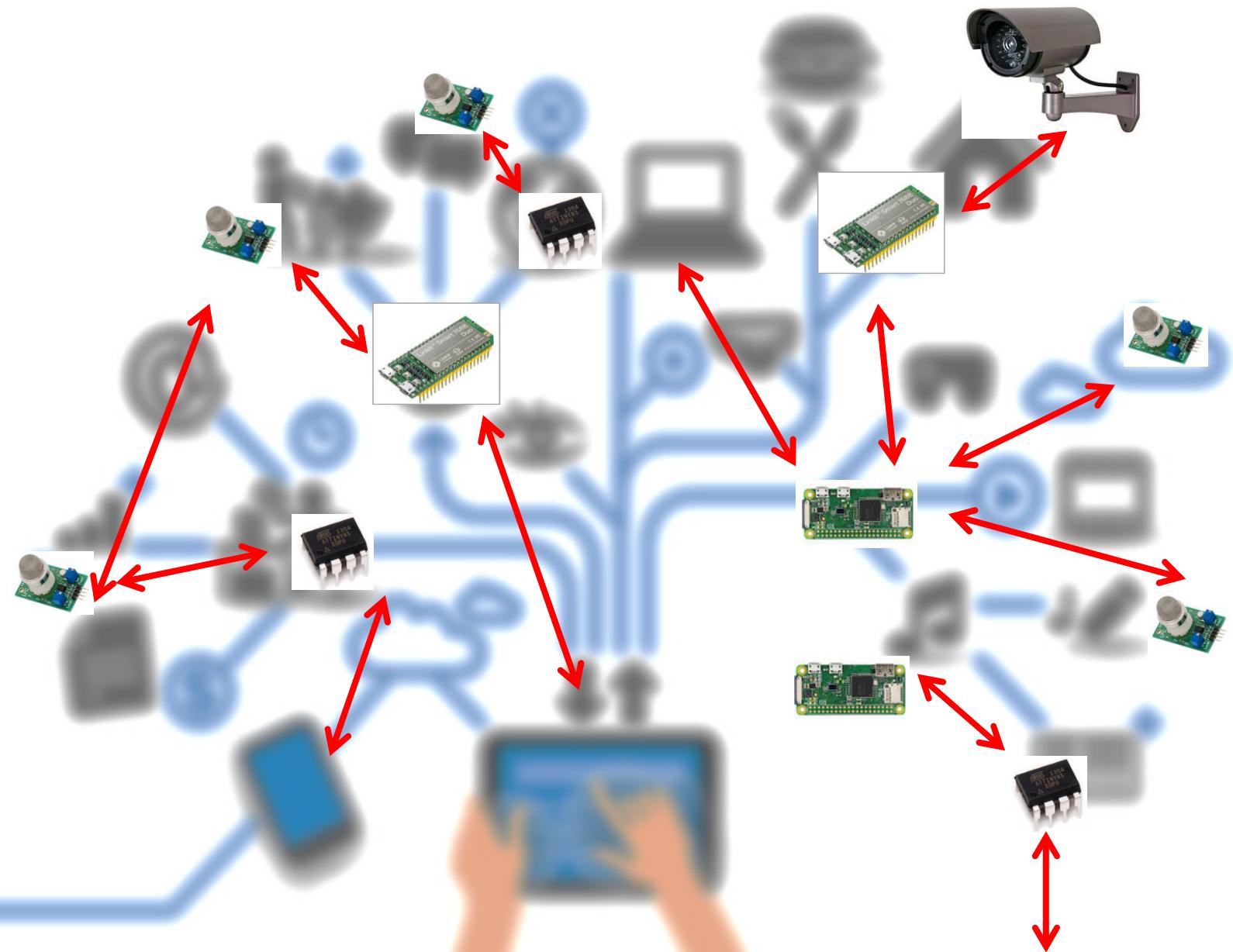
<https://www.inloox.fr/entreprise/blog>

Internet des objets (IOT *Internet Of Things*)

- Kevin Ashton (RFID) 1999
- *If we had computers that knew everything there was to know about things—using data they gathered without any help from us—we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best.*
- *We need to empower computers with their own means of gathering information, so they can see, hear and smell the world for themselves, in all its random glory.*



<https://www.inloox.fr/entreprise/blog>



<https://www.inloox.fr/entreprise/blog>



Internet des objets (IOT *Internet Of Things*)

- Système exploitant :
 - Réseau (& protocoles)
 - Capteurs
 - Micro contrôleurs et ordinateurs
 - Big Data
 - Intelligence Artificielle

Internet des objets (IOT *Internet Of Things*)

- Applications
 - Domaine de la santé
 - Surveillance (personnes, évènements, sécurité)
 - Pollution, temps, agriculture
 - Sécurité des objets – mesure d'usure et d'utilisation
 - Circulation – transport
 - Voiture
 - Gestion d'énergie
 - Maison, profils etc..
 - Gestion de stocks
 - Marketing / publicité
 - Education
 - Planification de ville
 - ...

Internet des objets (IOT *Internet Of Things*)

- Assistance et domotique
 - Gestion température
 - Les commandes frigo
 - Aide à la cuisine
 - Aide médicale
 - Surveillance et alerte
 - Détection d'anomalie et réparations
 - Sécurité

Internet des objets (IOT *Internet Of Things*)

- Capteurs / **effecteurs**
 - Température
 - Caméra
 - Radio
 - Microphone
 - Pulsations
 - Présence (IR / ultrason / pression)
 - Radiations
 - Gas
 - Position
 - Magnetometre
 - Gyroscope
 - Pression
 - Moteur
 - Electro aimants (contacteurs)
 - Résistances
 - Etc..
- Processeurs de données / IA
 - Ordinateur
 - Téléphone
 - Montre connectée
 - Micro contrôleur
 - Portes logiques électroniques
 - Relais
- Réseau
 - Internet
 - Ethernet
 - Radio (Lora etc..)
 - Wifi
 - Bluetooth
 - FM
 - Lumière
 - Sons
- Logiciel
 - Langages légers
 - Communication
 - IA
 - Protocoles réseau
 - Sécurité et crypto
 - Gestion Big Data
 - Fiabilité et redondance

Introduction



capteur



MCU



MPU + MCU



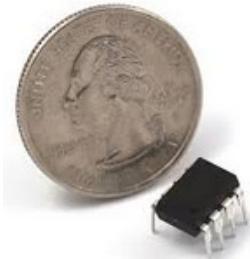
Micro-ordinateur
CPU + Mémoire + ... + ...



Introduction

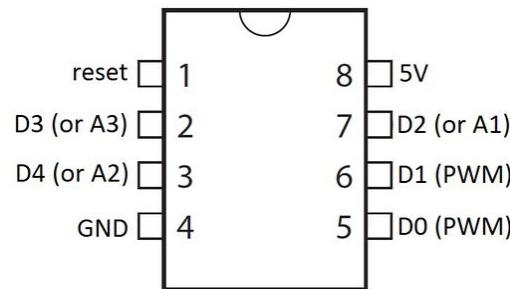
- Micro contrôleur : MCU
 - *Ordinateur* dédié à la gestion des capteurs
 - Intégré (mémoire etc..) : SOC System On Chip
 - Contrainte de taille
 - Consommation électrique
 - Coût de production
 - Mémoire limitée
 - Vitesse limitée
 - Accès électrique / électronique
 - ATTEL, STM32 etc..
 - Outils de développement spécialisés

Introduction



Attiny85

MCU



| Parameter Name | Value |
|-----------------------------------|--------------|
| Program Memory Type | Flash |
| Program Memory (KB) | 8 |
| CPU Speed (MIPS) | 20 |
| RAM Bytes | 512 |
| Data EEPROM (bytes) | 512 |
| Digital Communication Peripherals | 1-SPI, 1-I2C |
| Capture/Compare/PWM Peripherals | 5PWM |
| Timers | 2 x 8-bit |
| Comparators | 1 |
| Temperature Range (C) | -40 to 85 |
| Operating Voltage Range (V) | 1.8 to 5.5 |
| Pin Count | 8 |
| Cap Touch Channels | 3 |



10PCS Original **ATTINY85-20PU ATTINY85 20PU**
ATTINY85- 20 ATTINY85 DIP

Mega Semiconductor CO., Ltd.

US \$10.96 / lot

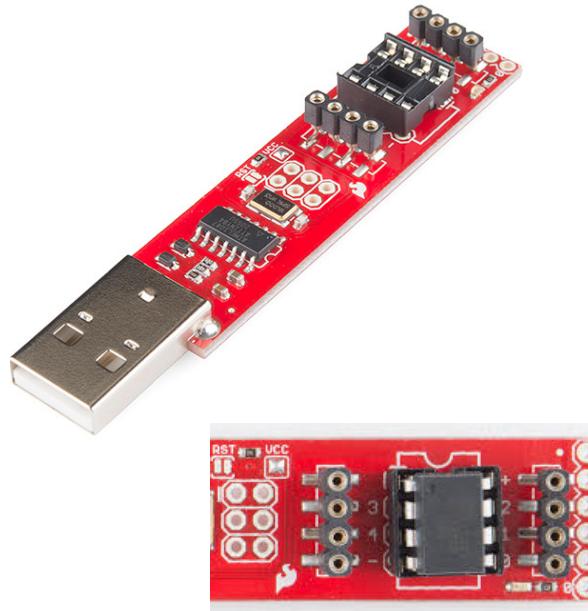
10 pieces / lot

Free Shipping

Feedback(324) | Orders (363)

Add to Wish List

Introduction



Programmation sur un pc
IDE : chargement ATTiny

```
37 // Initialize the OLED display using Wire library
38 SSD1306 display(0x3c, 4, 15);
39 // SH1106 display(0x3c, D3, D5);
40
41
42 #define DEMO_DURATION 3000
43 typedef void (*Demo)(void);
44
45 int demoMode = 0;
46 int counter = 1;
47
48 void setup() {
49     Serial.begin(115200);
50     Serial.println();
51     Serial.println();
52
53     pinMode(16, OUTPUT);
54     digitalWrite(16, LOW); // set GPIO16 low to reset OLE
55     delay(50);
56     digitalWrite(16, HIGH); // while OLED is running, mu
57     Wire.begin(4, 15);
58     display.init();
59
60
61     // Initialising the UI will init the display too.
62     display.init();
63
64     display.flipScreenVertically();
65     display.setFont(ArialMT_Plain_10);
66
67 }
68
```

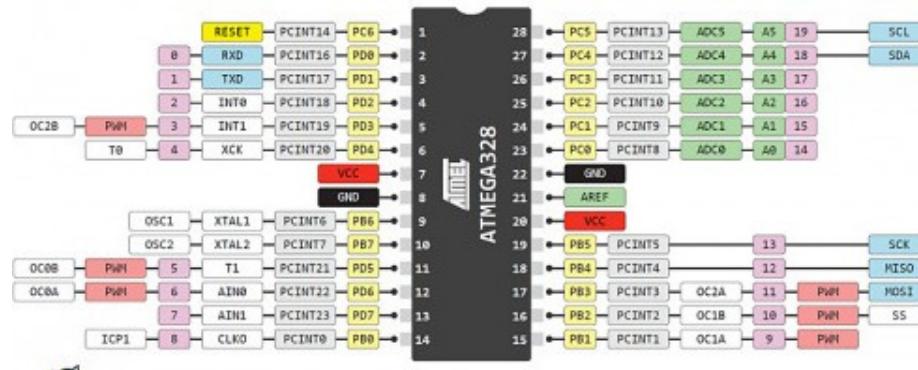
Introduction

atmega328



MCU

THE
DEFINITIVE
ATMEGA328
&Arduino
PINOUT DIAGRAM



Parameter Name

Value

Program Memory Type

Flash

Program Memory (KB)

32

CPU Speed (MIPS)

20

RAM Bytes

2,048

Data EEPROM (bytes)

1024

Digital Communication Peripherals

1-UART, 2-SPI, 1-I2C

Capture/Compare/PWM Peripherals

1 Input Capture, 1 CCP, 6PWM

Timers

2 x 8-bit, 1 x 16-bit

Comparators

1

Temperature Range (C)

-40 to 85

Operating Voltage Range (V)

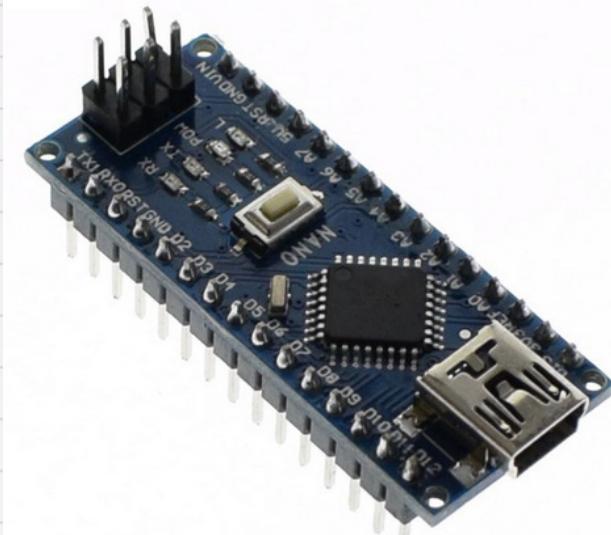
1.8 to 5.5

Pin Count

32

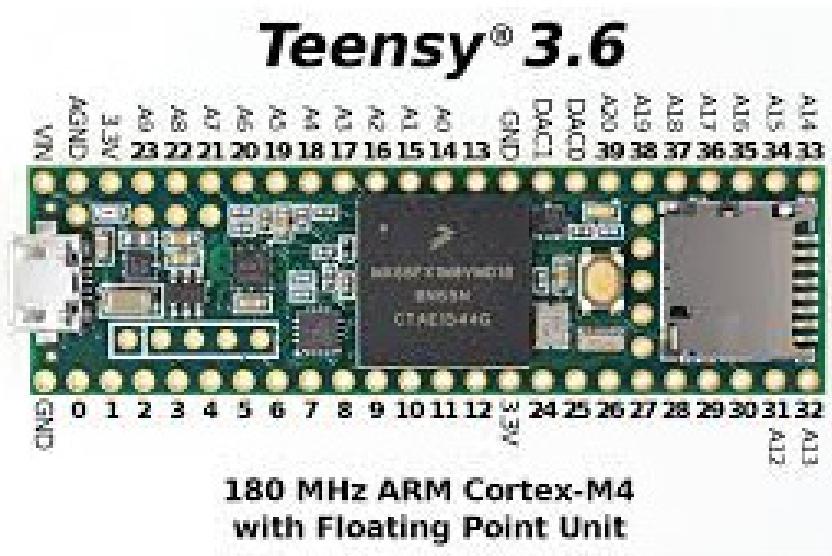
Cap Touch Channels

16



nano

Introduction

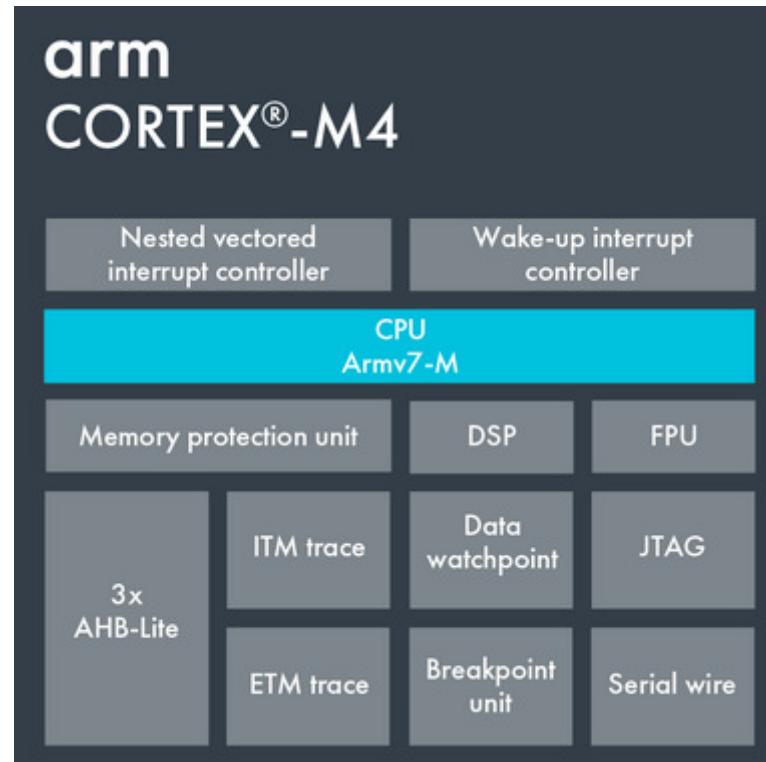


MPU + MCU

Prog en C

Mais communication ? périphérique

25 euros



STM32F429VIT6 - Microcontrôleur ARM, Haute Performance, STM32 F4 ARM Cortex-M4 Microcontroller, 180 MHz



| | |
|-------------------|----------------------------|
| Fabricant : | STMICROEL |
| Réf. Fabricant: | STM32F429VIT6 |
| Code Commande : | 2393659 |
| Gamme de produits | STM32 F4 A Microcontrol |

L'image a des fins d'illustration uniquement. Veuillez lire la description du produit.

[Ajouter au comparateur](#)[Rédiger Un Avis](#)

Aperçu du produit

The STM32F429VIT6 is a Microcontroller Unit, based on the high-performance ARM® Cortex®-M4 32-bit RISC core operating at a frequency of up to 180MHz. It features a floating point unit (FPU) single precision which supports all ARM® single-precision data-processing instructions and data types. It also implements memory protection instructions and a memory protection unit (MPU) which enhances application security. It incorporate high-speed embedded memories, up to 4Kbytes of flash and SRAM, an extensive range of enhanced I/Os and peripherals connected to two APB buses, two AHB buses and a 32-bit multi-AHB bus matrix.

- Up to 2MB of flash memory organized into two banks allowing read-while-write
- Up to 256+4kB SRAM including 64kB CCM data RAM
- LCD parallel interface, 8080/6800 modes
- Clock, reset and supply management
- 1.7 to 3.6V Application supply and I/Os
- Sleep, stop and standby modes
- 2x12-bit D/A Converters
- 16 Stream DMA controller with FIFOs and burst support
- SWD and JTAG interfaces
- Up to 168 I/O ports with interrupt capability
- 4 to 26MHz Crystal oscillator
- Internal 16MHz factory-trimmed RC (1% accuracy)
- 32kHz Oscillator for RTC with calibration
- Internal 32kHz RC with calibration
- Low-power
- Up to 21 communication interfaces
- Advanced connectivity
- 8 to 14-bit Parallel camera interface up to 54Mbytes/s
- True random number generator
- CRC calculation unit

Applications

Détection et Instrumentation, Electronique Grand Public

Introduction



MPU+ MCU + stockage 4 Mb+ communication Wifi

Caractéristiques [modifier | modifier le code]

L'ESP8266 est décliné en [plusieurs variantes](#) [archive]. Un exemple de caractéristiques est indiqué ci-dessous.

- 32-bit RISC CPU: Tensilica Xtensa LX106, 80 MHz
- 64 KiB of instruction RAM, 96 KiB of data RAM
- External QSPI flash - 512 KiB to 4 MiB (up to 16MiB is supported)
- IEEE 802.11 b/g/n Wi-Fi
 - Integrated TR switch, balun, LNA, power amplifier and matching network
 - WEP or WPA/WPA2 authentication, or open networks
- 16 GPIO pins
- SPI, I²C,
- I²S interfaces with DMA (sharing pins with GPIO)
- UART on dedicated pins, plus a transmit-only UART can be enabled on GPIO2
- 1 10-bit ADC

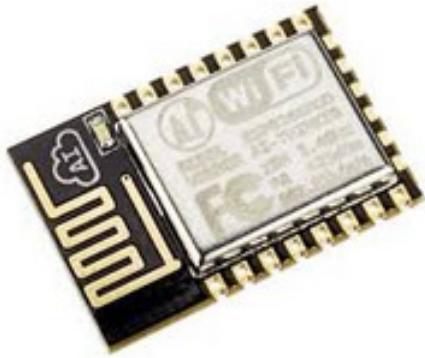
Avec régulation de tension + programmeur



WEMOS D1 mini Pro V1.1.0-16 M
octets externe antenne

US \$5.00 / pièce

Introduction



Prog C (espressif SDK)

NodeMCU : Lua

Langage interprété / primitives lien C

Stockage des programmes en mode texte

Mise à jour par wifi possible

Deep Sleep

Au lancement, une région de la flash ram = bootloader

Comment gérer l'execution ?

Comment stocker le code ?

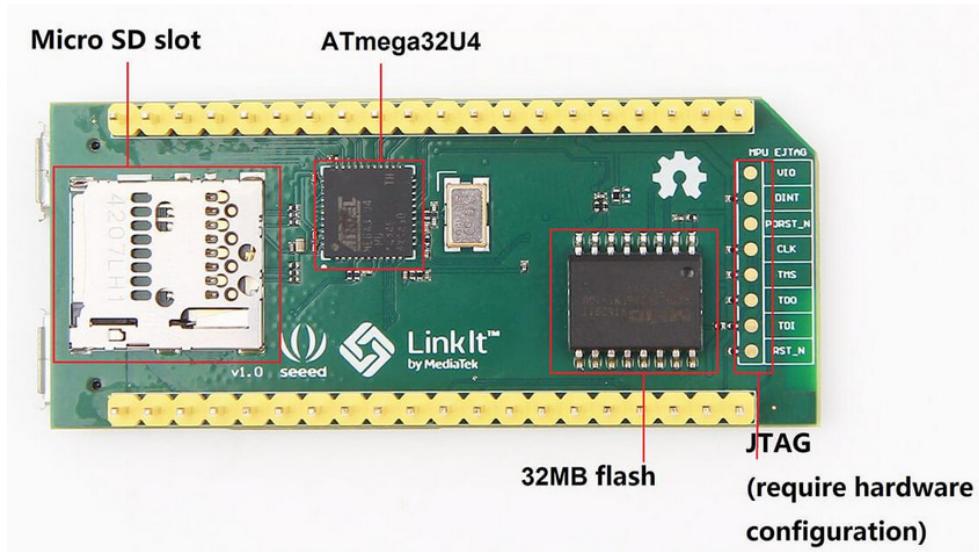
Etc..

ESP32 : evolution + LuaRtos OS (voir plus loin)

Introduction



- Single input single output(1T1R) Wi-Fi 802.11 b/g/n.
- Pin-out for GPIO, I2C, I2S, SPI, UART, PWM and Ethernet Port.
- 580 MHz MIPS CPU.
- 32MB flash and 128MB DDR2 RAM.
- USB host.
- Micro SD slot.

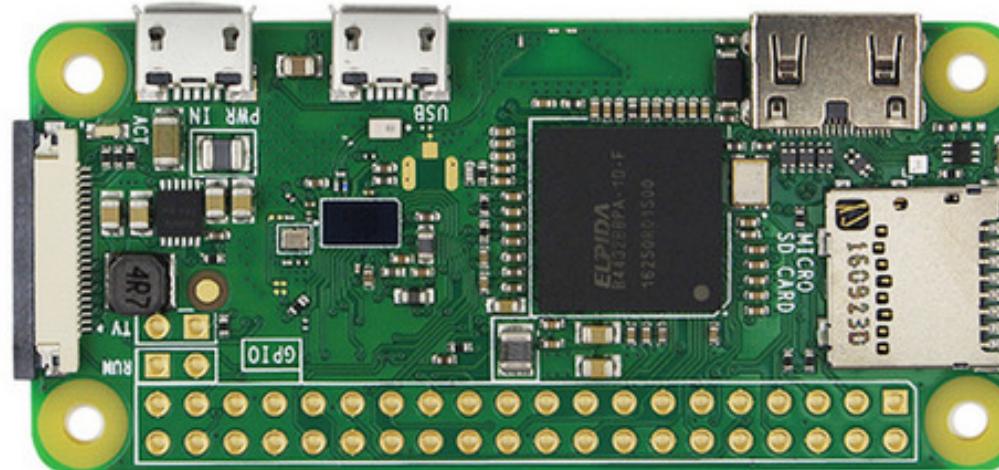


OpenWrt !
Linux embarqué (SSH, SAMBA..)
Wifi + Ethernet +

*node.js
python*

Introduction

Raspberry Pi Zero W



Machine linux

OS : moins réactif (threads etc) mais possibilité de calculs et de traitements

SD sensible aux coupures

Consommation plus importante

Maintenance et administration

Alimentation (2A)

Introduction

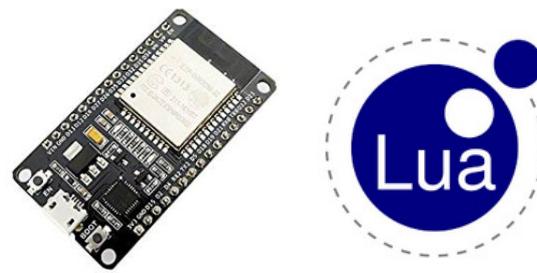


Processeur spécialisé

Deep Learning

IOT : ESP32

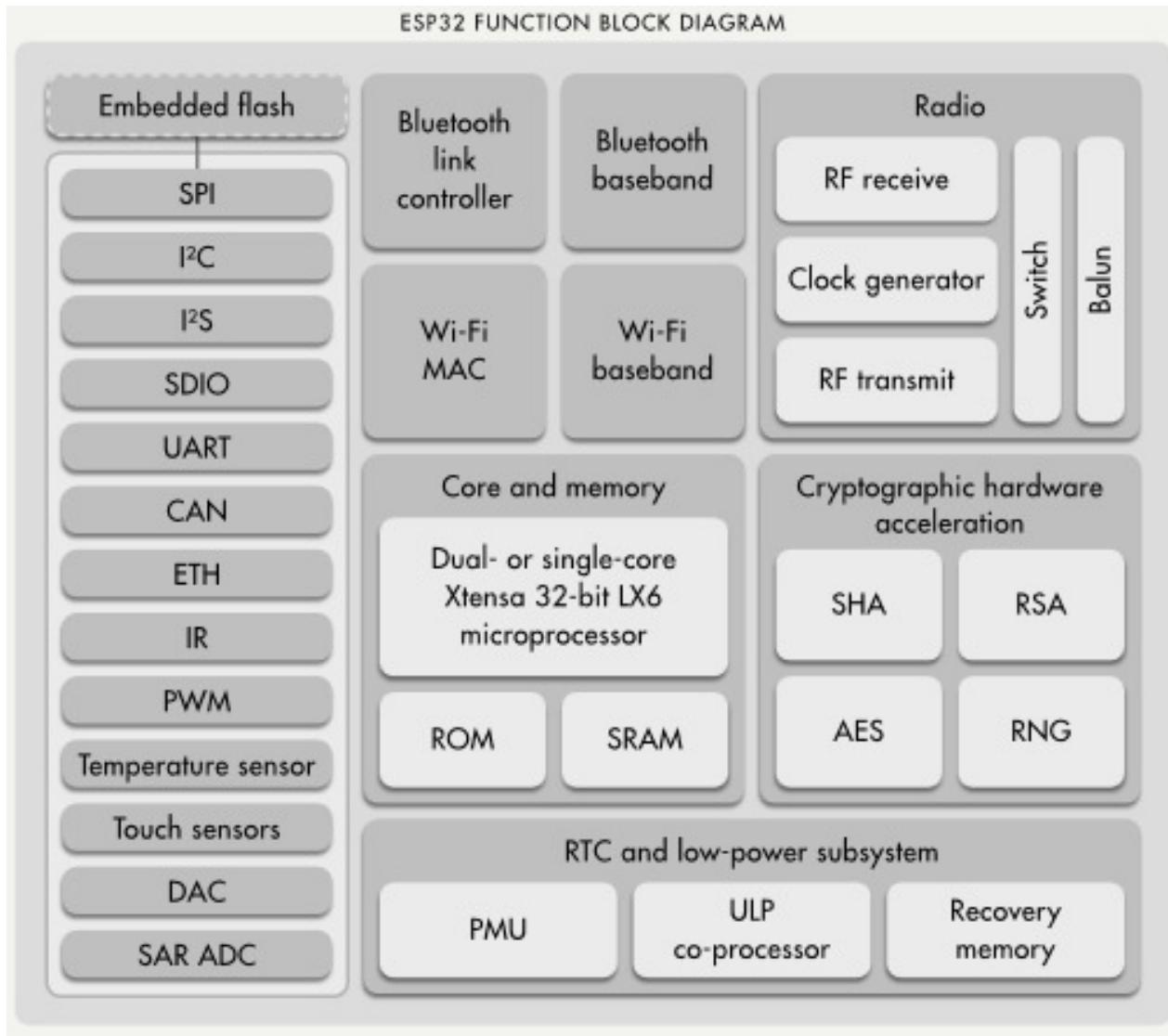
Initiation et programmation Lua



ESP32

- Tensilica Xtensa 32-bit LX6 microprocessor
 - **240 Mhz**
 - **600 DMIPS**
 - **Wi-Fi:** 802.11 b/g/n/e/i (802.11n @ 2.4 GHz up to 150 Mbit/s)
 - **Bluetooth:** v4.2 BR/EDR, Bluetooth Low Energy (BLE)
 - **Rom :** 448 KiB
 - **Sram :** 520 KiB
 - **RTC Slow + Fast Ram :** 8 KiB
 - **Embedded Flash :** 0 - 4 MiB
 - **External :** jusqu'à 16 MiB
 - **ADC, I2C, UART, CAN2, SPI, I2S, RMII, PWM etc..**
 - IEEE 802.11 : WFA, WPA/WPA2 et WAPI
 - Accélération matérielle : AES, SHA-2, RSA, elliptic curve cryptography (ECC), random number generator (RNG)
 - **Veille Deep Sleep 5 microA**

ESP32



+ Lora
NRF24
etc..

ESP32

- Développement
 - Espressif development framework (C)
 - Espruino (javascript)
 - DuckTape (javascript)
 - Moongoose OS (C et javascript)
 - mRuby
 - MicroPython
 - Zerynth
 - Arduino (processing)

Firmware
rom

ESP32

- Lua RTOS
 - Basé sur FreeRTOS
 - **Real Time Operating System**
 - Gestion du multitâches
- Lua
 - Langage fonctionnel
 - 1993 (Portugal : lune)
 - Très compact (180Ko) - rapide
 - C, C++
 - Lien simple avec les fonctions de bas niveau
 - Code source proche de Javascript



ESP32

- **Lua**

```
if condition1 and condition2 then
    -- instructions exécutées si les deux conditions sont vraies
elseif condition1 then
    -- instructions exécutées si la première condition est vraie
elseif condition2 then
    -- instructions exécutées si la deuxième condition est vraie
else
    -- instructions exécutées si les deux conditions sont fausses
end
```

```
function mafonction(arg1, arg2, ...)
    -- code
    return résultat1, résultat2
end
```

```
mafonction = function (arg1, arg2, ...)
    -- code
    return résultat1, résultat2
end
```

```
repeat
    -- code
until condition
```

```
while condition do
    -- code
end
```

```
for var = start, valend, step do
    -- code
end
```

```
for var_1, ..., var_n in explist do
    -- code
end
```

ESP32

- **Lua**

```
res1, res2 = mafonction(var)
```

```
do
  local mavariable = "chaîne"
  -- code utilisant mavariable
end
```

```
a = { 5, "foo", [[C:\Lua\Lua.exe]], 'bar', 42 }
```

```
a = { d = 5, [12] = "foo", ['chaîne avec espace'] = true }
```

ESP32

```
fonction = function (a, b) return (a + b) / a * b end
t =
{
    b =
    {
        -- Fonction comme clé
        [fonction] = 'Fonction !',
        -- Index numérique
        [5] = 42,
        -- Index chaîne simple (ie. syntaxe d'une variable)
        ls = [[Valeur
Multiligne]],
        -- Index chaîne quelconque
        [Expression rationnelle] = [[[?:\d{1,3}\.){3}\d{1,3}]],
        [ [[C:\Lua\Lua.exe]] ] = true,
    },
    -- Stockage d'une fonction
    f = function () return math.random(100) end,
    ff = fonction,
    -- Table comme clé
    [ { 'a', 'b' } ] = { "aa", "bb" },
}
-- Référence dans la table
t.reference = t.a
t[ { t.a, t.b } ] = t.f
```

ESP32

- Lua 5.3 <https://www.lua.org/manual/5.3/>



Lua 5.3 Reference Manual

The reference manual is the official definition of the Lua language.
For a complete introduction to Lua programming, see the book [Programming in Lua](#).

[start](#) · [contents](#) · [index](#) · [other versions](#)

Copyright © 2015–2017 Lua.org, PUC-Rio. Freely available under the terms of the [Lua license](#).

❖ Contents

[1 – Introduction](#)

[2 – Basic Concepts](#)

[2.1 – Values and Types](#)

[2.2 – Environments and the Global Environment](#)

[2.3 – Error Handling](#)

[2.4 – Metatables and Metamethods](#)

[2.5 – Garbage Collection](#)

[2.5.1 – Garbage-Collection Metamethods](#)

[2.5.2 – Weak Tables](#)

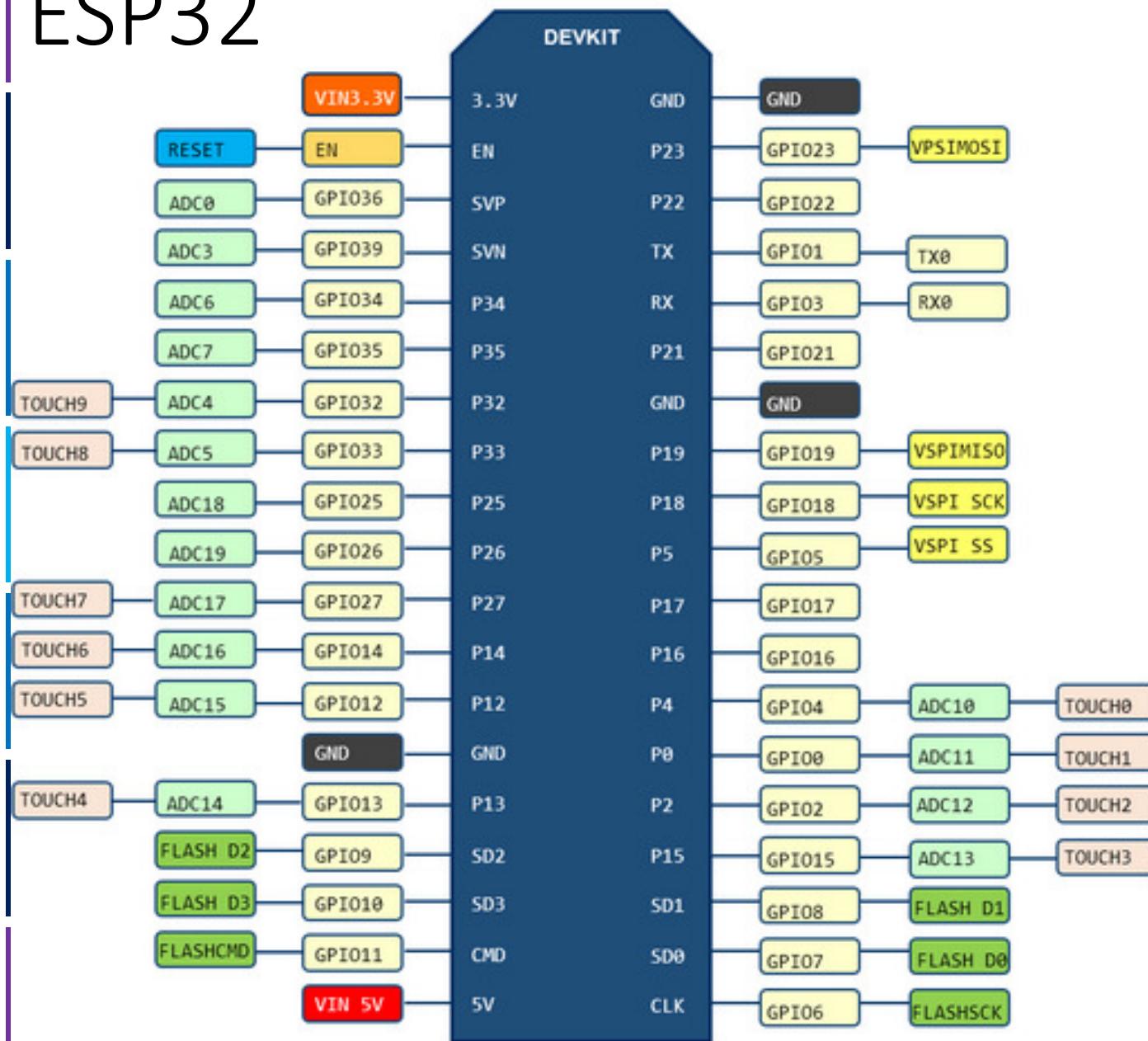
ESP32

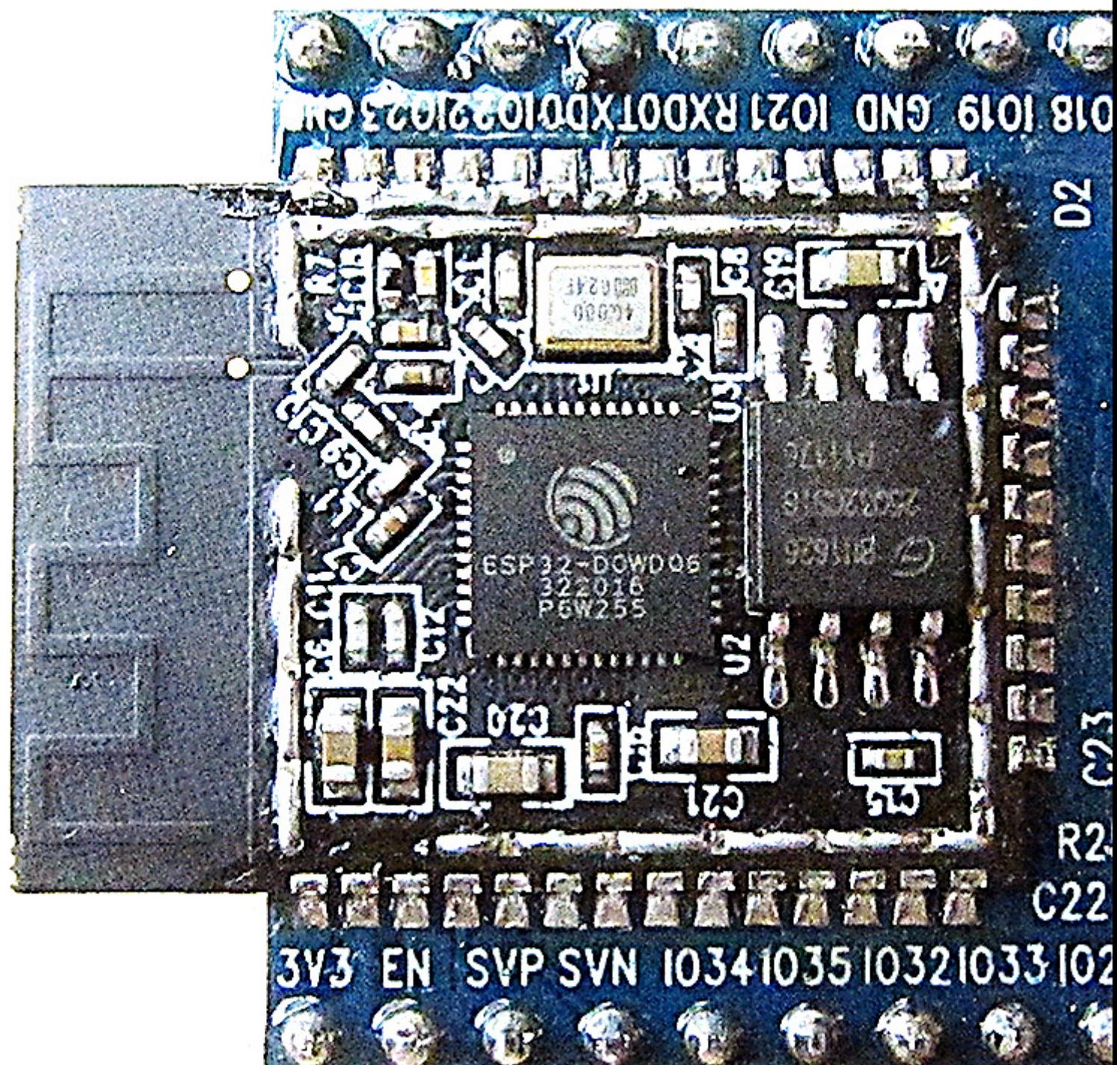
Esp32 WROOM 32



Exemple (plusieurs implantations)

ESP32





ESP32 + Lua RTOS

- **Attention**

- Vérifiez avant de brancher !!
- Essai / erreur = grosse bêtise
- Une seule erreur de branchement détruit la carte
- Définitivement
- En particulier, branchez toujours GND (ou -) en premier
- Branchez toujours le + (3v3) en dernier APRES avoir revérifié !
- Ne pas brancher ‘pour voir’ !!!
- Vérifier les contacts !!

Programmation

PC (windows / linux etc..)

Programmation du microcontrôleur

Cpu + mémoire flash etc..



Liaison USB
Liaison wifi / bluetooth
Radio
+
Alimentation 5v

exécute son propre code

```
4 Serial (COM)  
```

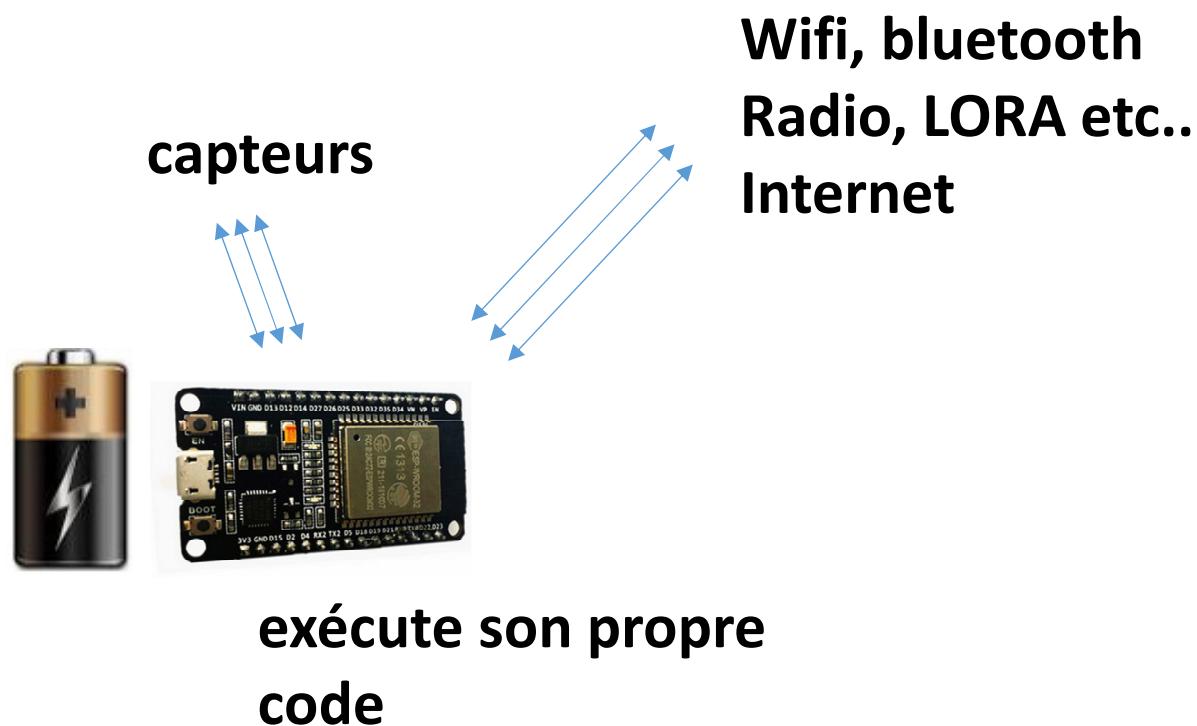
WHITE CAT

```
Lua RTOS beta 0.1. Copyright (C) 2015 - 2018 whitecatboard.org
build 1525396213
commit d418fc814d5d7e8743cd387df75790b913c8aed
Running from factory partition
board type ESP32THING
cpu ESP32 rev 0 at 240 Mhz
spiffs0 start address at 0x310000, size 512 Kb, partition spiffs
spiffs0 mounted

Lua RTOS beta 0.1 powered by Lua 5.3.4
Executing /system.lua ...
i2c0 at pins scl=GPIO23/sdc=GPIO22
IOT - Gildas Menier - Universite de Bretagne Sud
Executing /autorun.lua ...

/ > [ ]
```

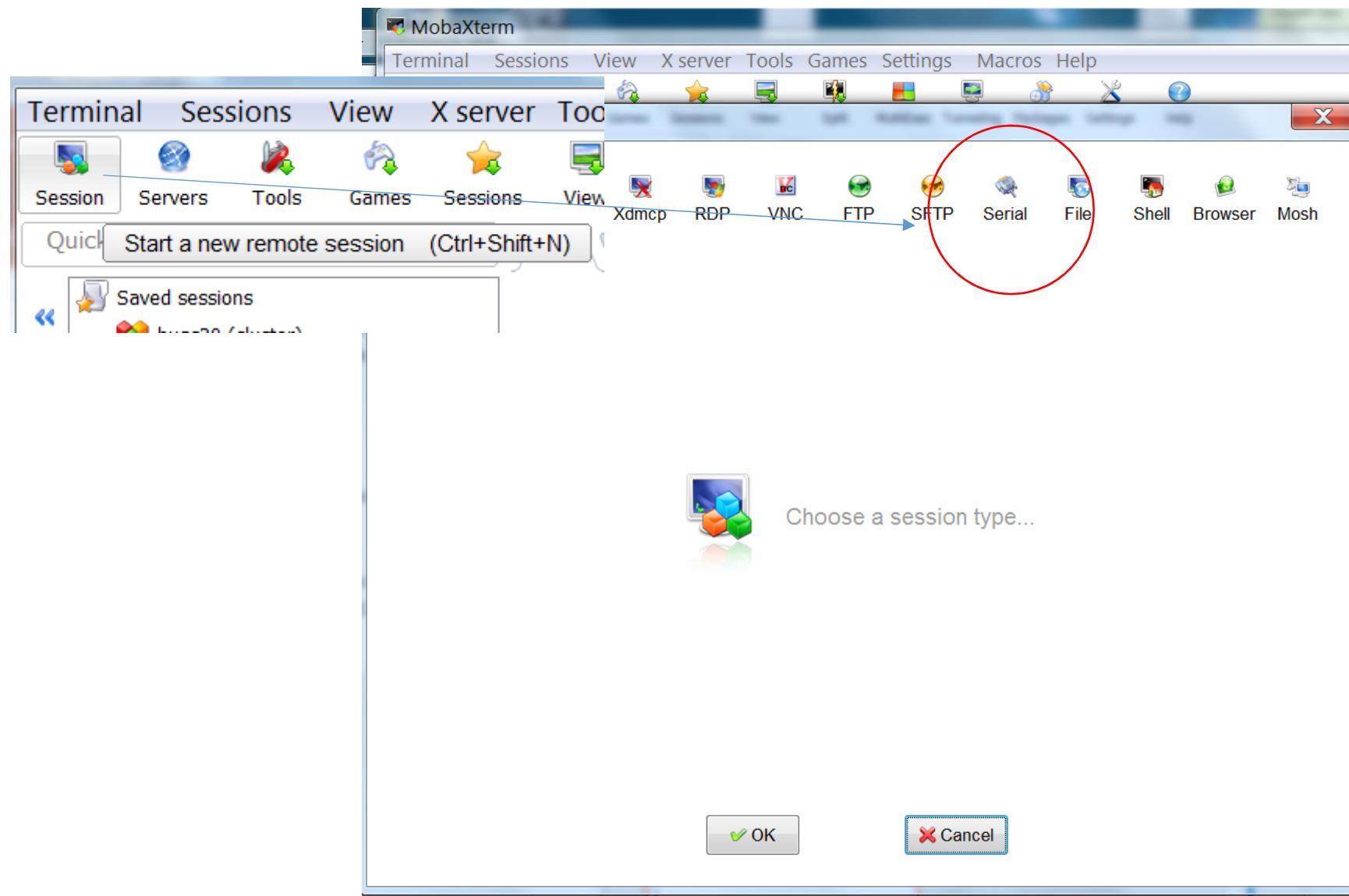
Autonomie



ESP32 + Lua RTOS

- **Environnement de programmation**
 - Le microcontrôleur se programme à l'aide d'une interface série
 - Branchement USB et installation driver
 - Console série
 - MobaXterm_Personal.exe

ESP32 + Lua RTOS





Basic Serial settings

Serial port * Choose at session start

Speed (bps) *

9600

1200

2400

4800

9600

19200

38400

57600

115200

460800

921600

Advanced Serial settings

Terminal settings

Bookmark settings

Puis ok
Et ouvrir la session : choisir COM

MobaXterm

Please choose your COM port for serial connection...



Serial port * COM10 (Lien série sur Bluetooth standard (COM10))

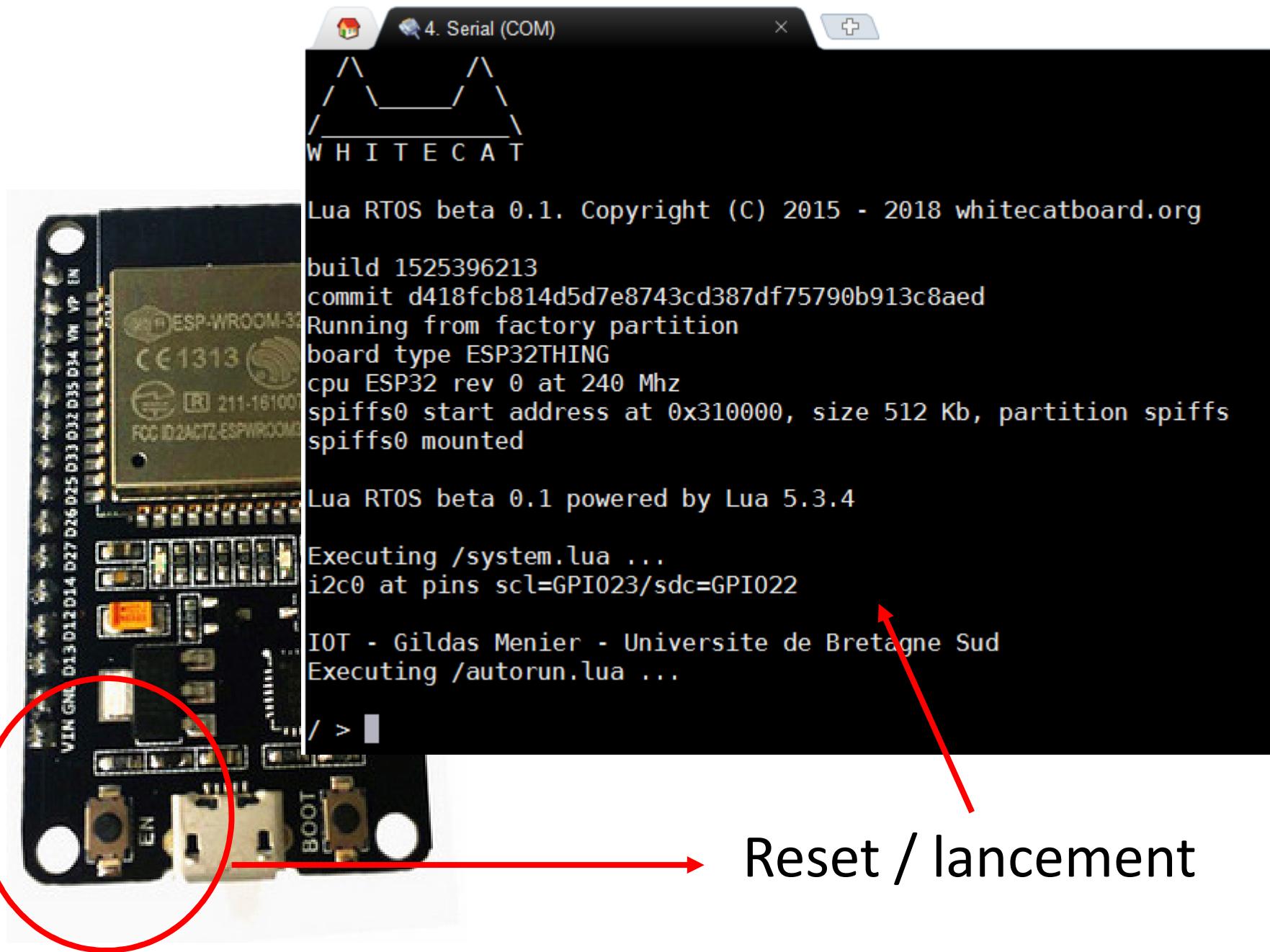
COM10 (Lien série sur Bluetooth standard (COM10))

COM12 (Silicon Labs CP210x USB to UART Bridge (COM12))

COM9 (Lien série sur Bluetooth standard (COM9))

COM3 (Intel(R) Active Management Technology - SOL (COM3))

OK



Reset / lancement

ESP32 + Lua RTOS

- **Console série**

- \$ picocom -b 115200 -d 8 -f n -p n /dev/ttyUSB0

en théorie facultatif



```
^__^
 \  \_____
   \    |
    )   (
   ^\   ^__)
  o   o=  )
        ||----w |
       ||     ||-----W H I T E C A T
       ||     ||-----^

Lua RTOS beta 0.1. Copyright (c) 2015 - 2017 whitecatboard.org

build 1506384326
commit ac70a1ba2177f26b16c74e84466a00f704158268
board type N1ESP32
cpu ESP32 rev 0 at 240 Mhz
flash EUI 304e363633037a11
spiffs0 start address at 0x180000, size 512 Kb
spiffs0 mounted

Lua RTOS beta 0.1 powered by Lua 5.3.4

Executing /system.lua ...
Executing /autorun.lua ...

/_run.lua:20: WARNING neopixel.setup is deprecated, please use neopixel.attach instead
/ > █
```

Rebooter bouton EN

CTRL-A CTRL-Q pour quitter ou CTRL-A CTRL-X

ESP32 + Lua RTOS

- **Ce n'est pas linux !**

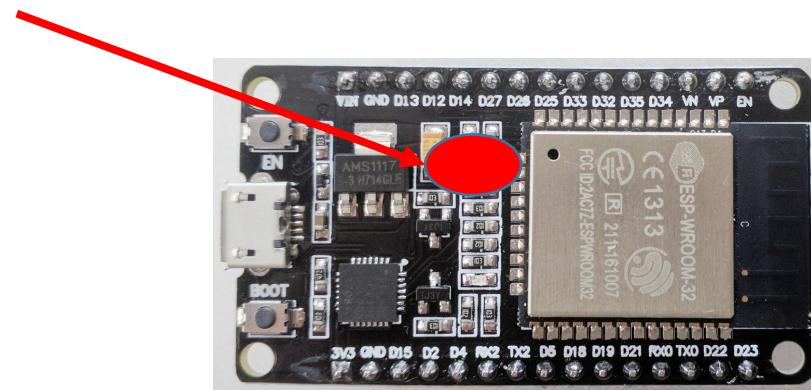
- **Ls**

- Pas de souris
- Pas de flèche haut
- CTRL C / CTRL V
- Etc..

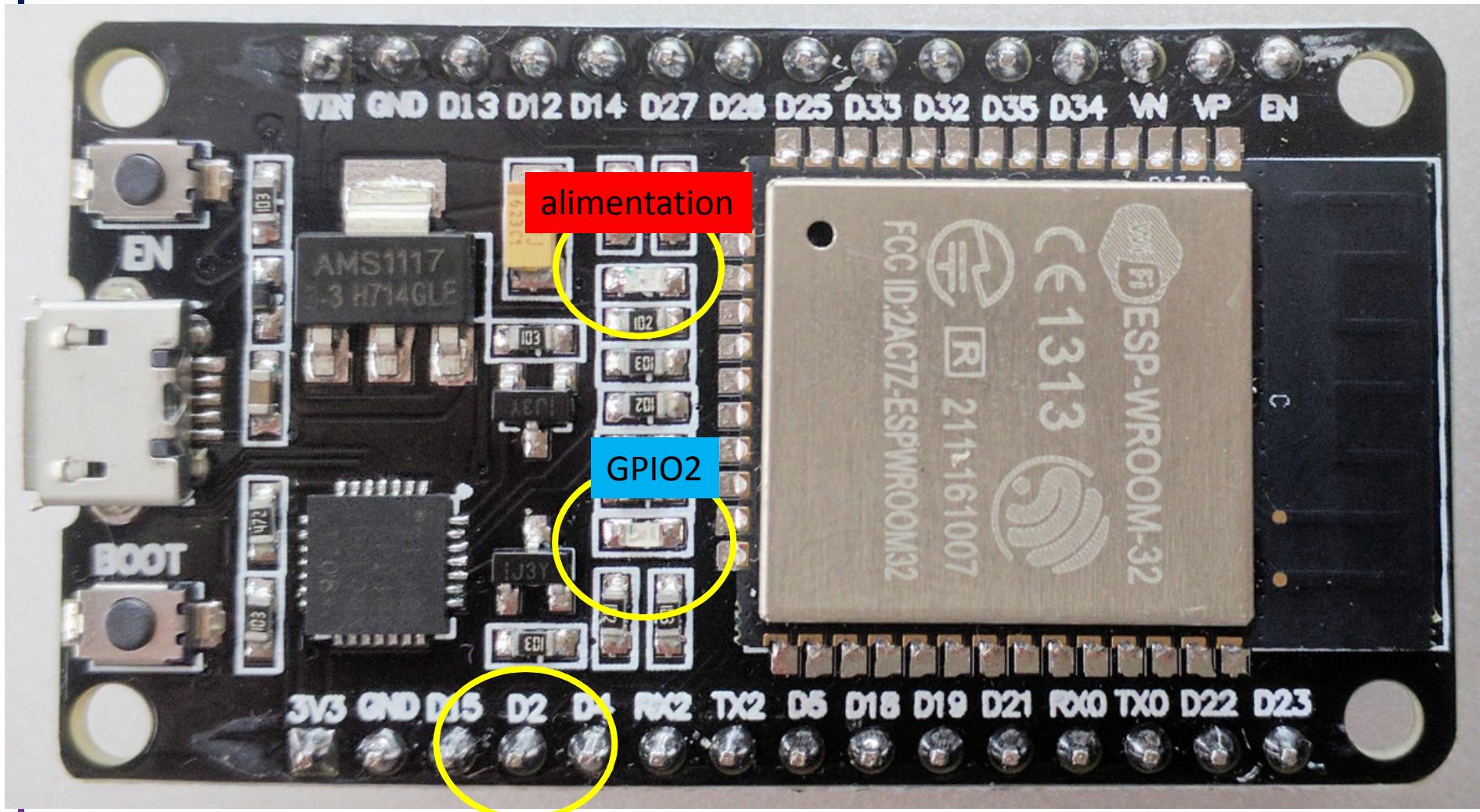
ESP32 + Lua RTOS

```
WHITE CAT
Lua RTOS beta 0.1. Copyright (c) 2015 - 2017 whitecatboard.org
build 15063241926
commit ac79a1ba217f726b16c74e84466a00f704158268
board type NIE5P32
cpu ESP32 rev 0 at 240 Mhz
flash EUI 394e363633837a11
flash_start address at 0x180000, size 512 Kb
spiffs0 mounted
Lua RTOS beta 0.1 powered by Lua 5.3.4
Executing /system.lua ...
Executing /autorun.lua ...
/run.lua:20: WARNING neopixel.setup is deprecated, please use neopixel.attach instead
/> █
```

- Les pattes de l'ESP32 sont des **GPIO**
 - *General Purpose Input Output*
 - On peut indiquer qu'on souhaite changer leur tension (0v ou 3v)
 - On peut indiquer qu'on souhaite lire leur tension
 - Lumière rouge : sous tension



ESP32



ESP32 + Lua RTOS

Vous allez forcer la tension de la patte (pin) 2 et allumer la led bleu

pio.pin.setdir(pio.OUTPUT, pio.GPIO2)

on indique que cette patte GPIO2 sert de sortie (output)

pio.pin.setpull(pio.NOPULL, pio.GPIO2)

pull up / désactivé

pio.pin.sethigh(pio.GPIO2)

on force le niveau de la patte 2 à 3v (la led doit s'allumer)

high est l'état : == 1 ou 3.3v

Attention, certains micro-contrôleurs 5v

ESP32 + Lua RTOS

Éteignez la led

pio.pin.setlow(pio.GPIO2)

on force le niveau de la patte 2 à 0v (la led doit s'éteindre)

Même chose que pio.pin.setval(0, pio.GPIO2)

ESP32 + Lua RTOS

Deux fonctions lua

ledon() : allumer la led interne

ledoff() : l'éteindre

ESP32

- Lua

```
function mafonction(arg1, arg2, ...)  
    -- code  
    return résultat1, résultat2  
end
```

```
mafonction = function (arg1, arg2, ...)  
    -- code  
    return résultat1, résultat2  
end
```

```
pio.pin.setdir(pio.OUTPUT, pio.GPIO2)
```

```
pio.pin.sethigh(pio.GPIO2)
```

ESP32 + Lua RTOS

Remarque : ledon, ledoff

```
ledon = function()
    pio.pin.setdir(pio.OUTPUT, pio.GPIO2)
    pio.pin.sethigh(pio.GPIO2)
end
```

Ou encore, c'est ok d'écrire :

```
function ledon()
    pio.pin.setdir(pio.OUTPUT, pio.GPIO2)
    pio.pin.sethigh(pio.GPIO2)
end
```

ESP32 + Lua RTOS

Boucle Lua

```
/ > for j=1,5 do ledon(); tmr.delay(1); ledoff(); tmr.delay(1); end  
/ > █
```

Ou bien

```
/ > for j=1,5 do  
=> ledon();  
=> tmr.delay(1)  
=> ledoff();  
=> tmr.delay(1)  
=> end  
/ > █
```

Attention, pas de correction possible

tmr.delay() est une fonction de la librairie tmr (ou timer)

delay est en secondes

tmr.delayms() en millisecondes

ESP32 + Lua RTOS

Fonction Lua

```
/> hello = function() print("bonjour") end  
/> hello()  
bonjour  
/> █
```

Ou bien :

```
/>  
/> hello = function()  
->     print("bonjour")  
-> end  
/> hello()  
bonjour  
/> █
```

ESP32 + Lua RTOS

Fonction Lua :

réalisez une fonction ‘blink’ qui fait clignoter 3 fois la lumière

ESP32 + Lua RTOS

Fonction Lua :

```
/> blink = function()
>> for j=1,3 do
>>     ledon(); tmr.delayms(200); ledoff(); tmr.delayms(200);
>> end
>> end
/> blink()
/> █
```

```
blink = function()
    for j=1,3 do
        ledon(); tmr.delayms(200);
        ledoff(); tmr.delayms(200);
    end
end
```

Attention,
blink est la fonction
blink() est l'execution de la fonction

ESP32 + Lua RTOS

Fonction Lua : blink avec le nombre de clignotement

```
blink = function(nb)
    for j=1,nb do
        ledon(); tmr.delayms(200);
        ledoff(); tmr.delayms(200);
    end
end
```

ESP32 + Lua RTOS

Fonction Lua : blink avec le nombre de clignotement

```
function blink(nb)
    for j=1,nb do
        ledon(); tmr.delayms(200);
        ledoff(); tmr.delayms(200);
    end
end
```

ESP32 + Lua RTOS

Boucle lua

```
while true do
    ledon(); tmr.delayms(50);
    ledoff(); tmr.delayms(50);
end
```

Essayez... remarque ?

CTRL-C !

```
/ > while true do
>>     ledon(); tmr.delayms(50);
>>     ledoff(); tmr.delayms(50);
>> end
stdin:2: interrupted!
stack traceback:
[C]: in field 'delayms'
stdin:2: in main chunk
[C]: in ?

/ >
/ >
/ > █
```

ESP32 + Lua RTOS

Boucle lua et fonction

```
function clignote()
    while true do
        ledon(); tmr.delayms(50);
        ledoff(); tmr.delayms(50);
    end
end
```

CTRL-C également

```
/ > clignote
function: 0x3ffebf00
/ > clignote()
stdin:3: interrupted!
stack traceback:
[C]: in field 'delayms'
stdin:3: in function 'clignote'
(...tail calls...)
[C]: in ?
/ >
/ >
/ > █
```

ESP32 + Lua RTOS

Boucle lua et fonction

En fait, en Internet des objets, on peut avoir un objet qui fait toujours la même chose sans arrêt (sauf si on le débranche) !

Pas de problème pour la boucle infinie, c'est ok !

Le microcontrôleur se lance et exécute la boucle infinie

ESP32 mini système d'exploitation temps réel :

RTOS : Real Time Operating System

ESP32 + Lua RTOS

Thread

Comme en Java, on peut partager le temps processeur entre plusieurs exécutions

Thread RTOS

Un thread est un fil d'exécution

L'ESP32 possède deux cœurs pour l'interprétation de LUA

LUA RTOS

ESP32 + Lua RTOS

`thread.list()`

```
/ >
/ > thread.list()
-----
```

| THID | TYPE | NAME | STATUS | CORE | PRI0 | SIZE | STACK FREE | USED |
|------------|------|----------|--------|------|------|-------|------------|-------------|
| 1073602752 | lua | lua_main | run | 0 | 20 | 10240 | 6164 | 4076 (39%) |

```
/ > █
```

`thread.list(true)`

```
/ > thread.list(true)
table: 0x3ffdा5e4
```

Un seul thread fonctionne : la boucle d'interpréteur principal

Le while infini bloque cette boucle

ESP32 + Lua RTOS

Création d'un thread pour la boucle infinie

t = thread.start(clignote)

... fabrique un nouveau thread qui fonctionne en même temps que le thread principal :

| thread.list() | | | | | | | | |
|---------------|------|------------|--------|------|------|-------|------------|-------------|
| THID | TYPE | NAME | STATUS | CORE | PRI0 | SIZE | STACK FREE | USED |
| 1073602752 | lua | lua_main | run | 0 | 20 | 10240 | 6164 | 4076 (39%) |
| 1073655936 | lua | lua_thread | run | 1 | 20 | 8192 | 7028 | 1164 (14%) |
| / > █ | | | | | | | | |

Ne bloque pas les commandes

La led continue de clignoter (CTRL-C n'est plus possible)

ESP32 + Lua RTOS

Pause d'un thread

thread.suspend(t)

... mise en pause du thread t

```
/ > thread.suspend(t)
/ > thread.list()
```

| THID | TYPE | NAME | STATUS | CORE | PRI0 | SIZE | STACK FREE | USED |
|------------|------|------------|--------|------|------|-------|------------|-------------|
| 1073602752 | lua | lua_main | susp | 0 | 20 | 10240 | 6164 | 4076 (39%) |
| 1073655936 | lua | lua_thread | run | 1 | 20 | 8192 | 7028 | 1164 (14%) |

```
/ > █
```

La led ne clignote plus

ESP32 + Lua RTOS

Pause d'un thread

thread.resume(t)

... redémarrage

| / > thread.resume(t) / > thread.list() | THID | TYPE | NAME | STATUS | CORE | PRI0 | SIZE | STACK FREE | USED |
|---|------------|------|------------|--------|------|------|-------|------------|-------------|
| | 1073602752 | lua | lua_main | run | 0 | 20 | 10240 | 6164 | 4076 (39%) |
| | 1073655936 | lua | lua_thread | run | 1 | 20 | 8192 | 7028 | 1164 (14%) |
| / > █ | | | | | | | | | |

La led se remet à clignoter

ESP32 + Lua RTOS

Pause d'un thread

thread.stop(t)

Arrêt et vidage mémoire

| / > thread.stop(t) | / > thread.list() | | | | | | | |
|--------------------|-------------------|----------|--------|------|------|-------|------------|-------------|
| THID | TYPE | NAME | STATUS | CORE | PRI0 | SIZE | STACK FREE | USED |
| 1073602752 | lua | lua_main | run | 0 | 20 | 10240 | 6164 | 4076 (39%) |
| / > | █ | | | | | | | |

La led reste en l'état

- Console
- Ensemble de commandes
 - Ce n'est pas linux !
 - Pseudo Shell
 - Interpréteur Lua
 - Commandes simplifiées : accès aux 4 Mo de stockage

- cat

- cd

- clear

- cp

- reboot

- ls

- luac

- mkdir

- more

- mv

- pwd

- rm

Attention : n'effacez ou ne modifiez rien
sous peine de ne plus pouvoir faire le TD
Pas de protection particulière

**On suppose que l'utilisateur sait ce qu'il fait
(?!)**

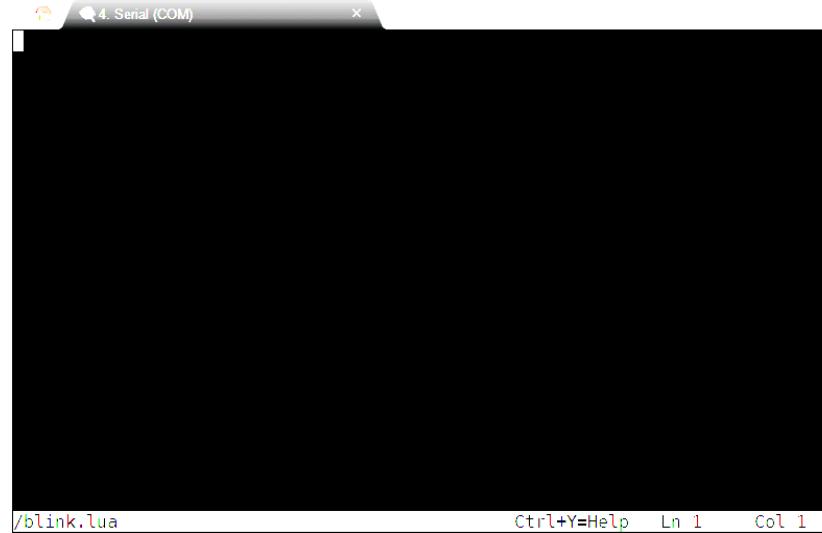
ESP32 + Lua RTOS

- reboot
- Relance le microcontrôleur
- Toutes les définitions précédentes sont perdues
- Il est possible de créer un fichier et de l'exécuter
- Attention, le fichier se trouve sur le microcontrôleur
 - PAS SUR LE PC
- Réalisez un fichier qui contient la définition de blink(n)
- Puis qui fait clignoter 3 fois la led

```
/ > edit blink.lua
```

ESP32 + Lua RTOS

- Mini éditeur de texte
- Attention, c'est une liaison série, ce n'est pas un programme PC
- Communication seulement par texte avec le microcontrôleur
- Pas de souris, pas de CTRL-C CTRL-V
- Curseur
- CTRL-Y : aide
- CTRL-S: sauvegarde
- CTRL-Q : quitter



Editor Command Summary

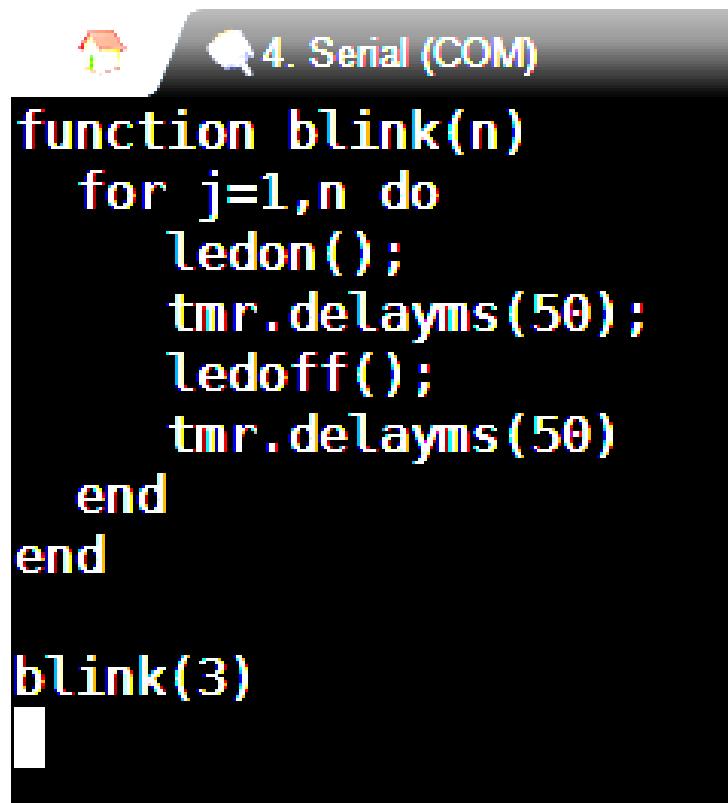
| | | | |
|--------------|------------------------------|--------|-----------------------------|
| <up> | Move one line up (*) | Ctrl+N | New editor |
| <down> | Move one line down (*) | Ctrl+O | Open file |
| <left> | Move one character left (*) | Ctrl+S | Save file |
| <right> | Move one character right (*) | Ctrl+W | Close file |
| <pgup> | Move one page up (*) | Ctrl+Q | Quit |
| <pgdn> | Move one page down (*) | Ctrl+P | Pipe command |
| Ctrl+<left> | Move to previous word (*) | Ctrl+A | Select all |
| Ctrl+<right> | Move to next word (*) | Ctrl+C | Copy selection to clipboard |
| <home> | Move to start of line (*) | Ctrl+X | Cut selection to clipboard |
| <end> | Move to end of line (*) | Ctrl+V | Paste from clipboard |
| Ctrl+<home> | Move to start of file (*) | Ctrl+Z | Undo |
| Ctrl+<end> | Move to end of file (*) | Ctrl+R | Redo |
| <backspace> | Delete previous character | Ctrl+F | Find text |
| <delete> | Delete current character | Ctrl+G | Find next |
| Ctrl+<tab> | Next editor | Ctrl+L | Goto line |
| <tab> | Indent selection | Ctrl+Y | Help |
| Shift+<tab> | Unindent selection | | |

(*) Extends selection if combined with Shift

Press any key to continue... █

ESP32 + Lua RTOS

- Mini éditeur de texte



```
function blink(n)
    for j=1,n do
        ledon();
        tmr.delayms(50);
        ledoff();
        tmr.delayms(50)
    end
end

blink(3)
```

CTRL-S : sauvegarder
CTRL-Q : quitter éditeur

```
/ > ls
d      .
d      ..
d      examples
d      www
f      5169   _info.lua
f      0       _run.lua
f      2       autorun.lua
f      2927   console.lua
f      2538   system.lua
f      2409   config.lua
f      100    blink.lua
/ > █
```

ESP32 + Lua RTOS

Remarque : ledon, ledoff

```
ledon = function()
    pio.pin.setdir(pio.OUTPUT, pio.GPIO2)
    pio.pin.sethigh(pio.GPIO2)
end
```

Ou encore, c'est ok d'écrire :

```
function ledon()
    pio.pin.setdir(pio.OUTPUT, pio.GPIO2)
    pio.pin.sethigh(pio.GPIO2)
end
```

ESP32 + Lua RTOS

- Le fichier a été créé :

```
/ > cat blink.lua
function blink(n)
    for j=1,n do
        ledon();
        tmr.delayms(50);
        ledoff();
        tmr.delayms(50)
    end
end

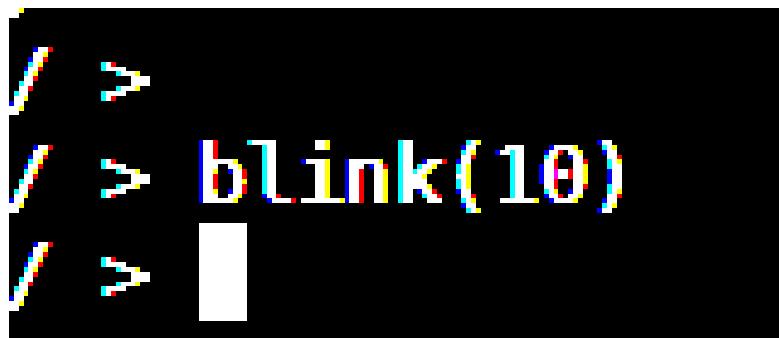
blink(3)
/ > █
```

Mais n'est pas encore interprété :

```
/ > blink(3)
stdin:1: attempt to call a nil value (global 'blink')
stack traceback:
    stdin:1: in main chunk
    [C]: in ?
/ > █
```

ESP32 + Lua RTOS

- Lancement de l'exécution du fichier **blink.lua** :
- **dofile("blink.lua")**
- La console bloque le temps de l'exécution
- Même thread que le système principal
- Maintenant la fonction **blink** est définie :

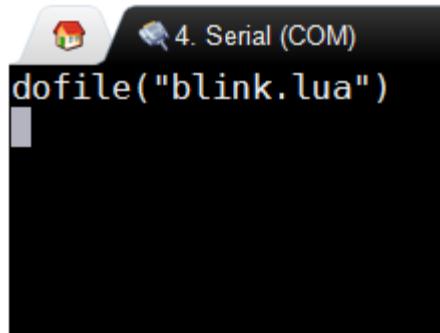


ESP32 + Lua RTOS

- Les fichiers sont conservés après un redémarrage
- Faire ‘ls’
- Faire ‘reboot’
- Faire ‘ls’
- Les fichiers sont préservés
- Mémoire Flash de 4Mo
- C'est peu mais ne pas oublier que ce n'est pas un ordinateur, c'est un microcontrôleur !

ESP32 + Lua RTOS

- On peut lancer un script directement au démarrage
- Créer un fichier **autorun.lua**
- edit **autorun.lua**



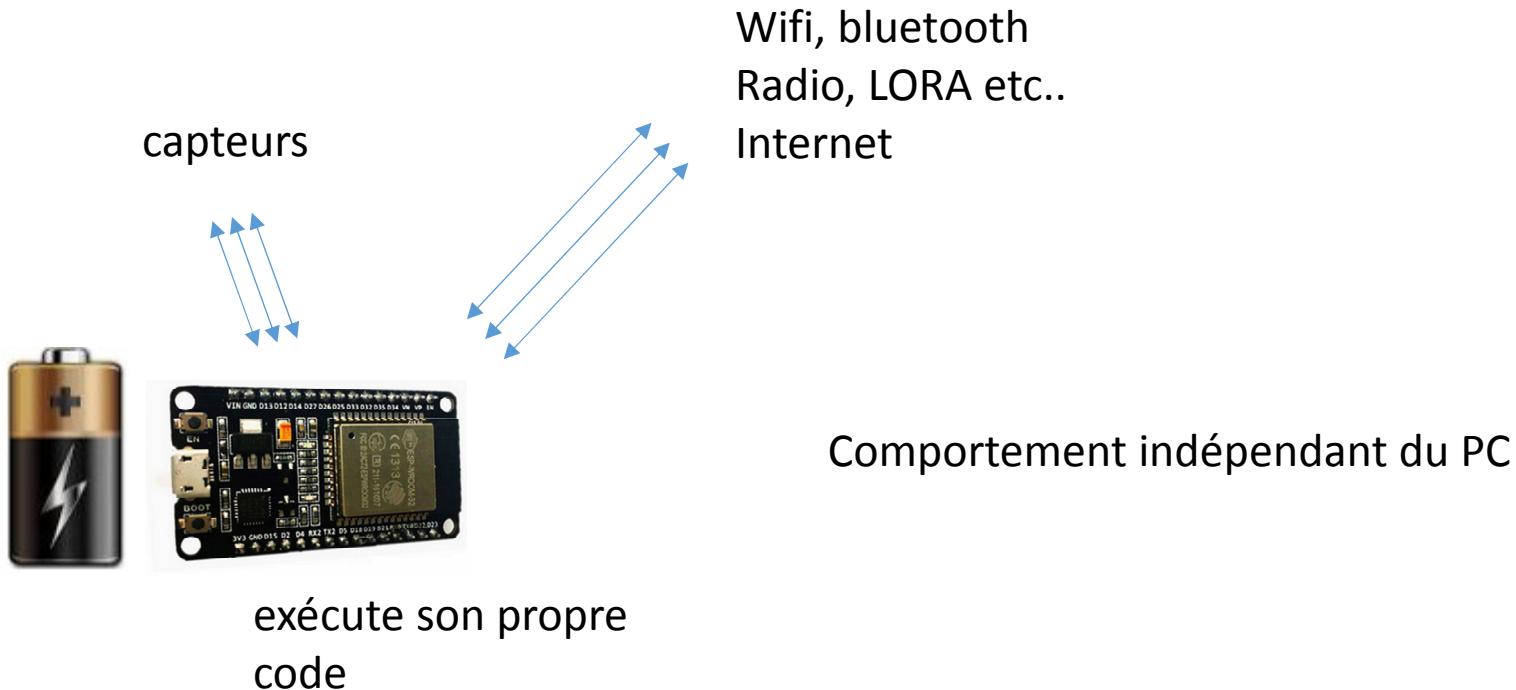
- CTRL-S puis CTRL-Q
- reboot
- Vérifier que la led clignote au démarrage

Autonomie

Le microcontrôleur est autonome une fois programmé

Ici, vous avez programmé le microcontrôleur pour qu'il fasse clignoter la led au démarrage

Testez avec une batterie SANS la liaison usb ordinateur



ESP32 + Lua RTOS

Modifiez le fichier autorun.lua pour y mettre :

```
print("Hello !!!")
```

```
dofile("blink.lua")
```

```
blink(5)
```

```
print("Ca va ?")
```

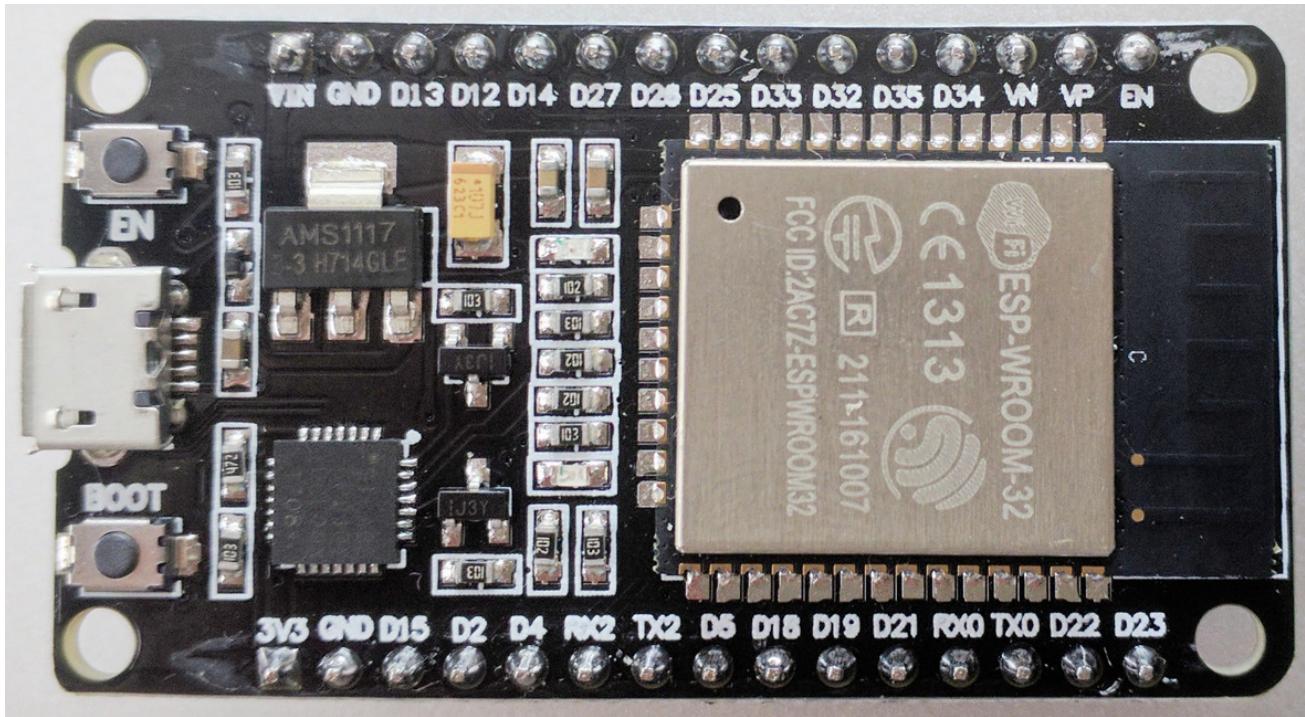
Que fait l'esp32 au démarrage ?

ESP32 + Lua RTOS

L'esp32 est équipé de connecteurs

Les pattes notées GND sont les masses (polarité -3v)

Les pattes notées 3V3 sont les + 3.3v

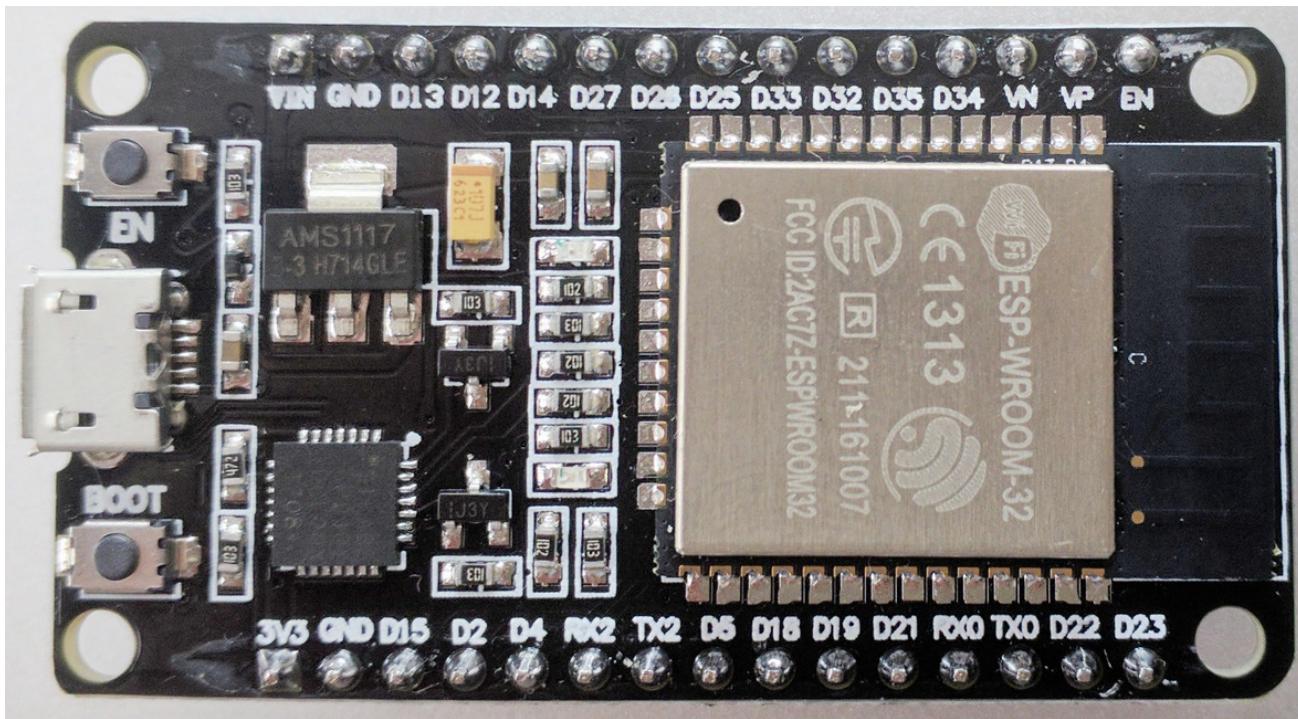


ESP32 + Lua RTOS

Les pattes notées TX envoient des caractères texte

Les pattes notées RX peuvent recevoir du texte

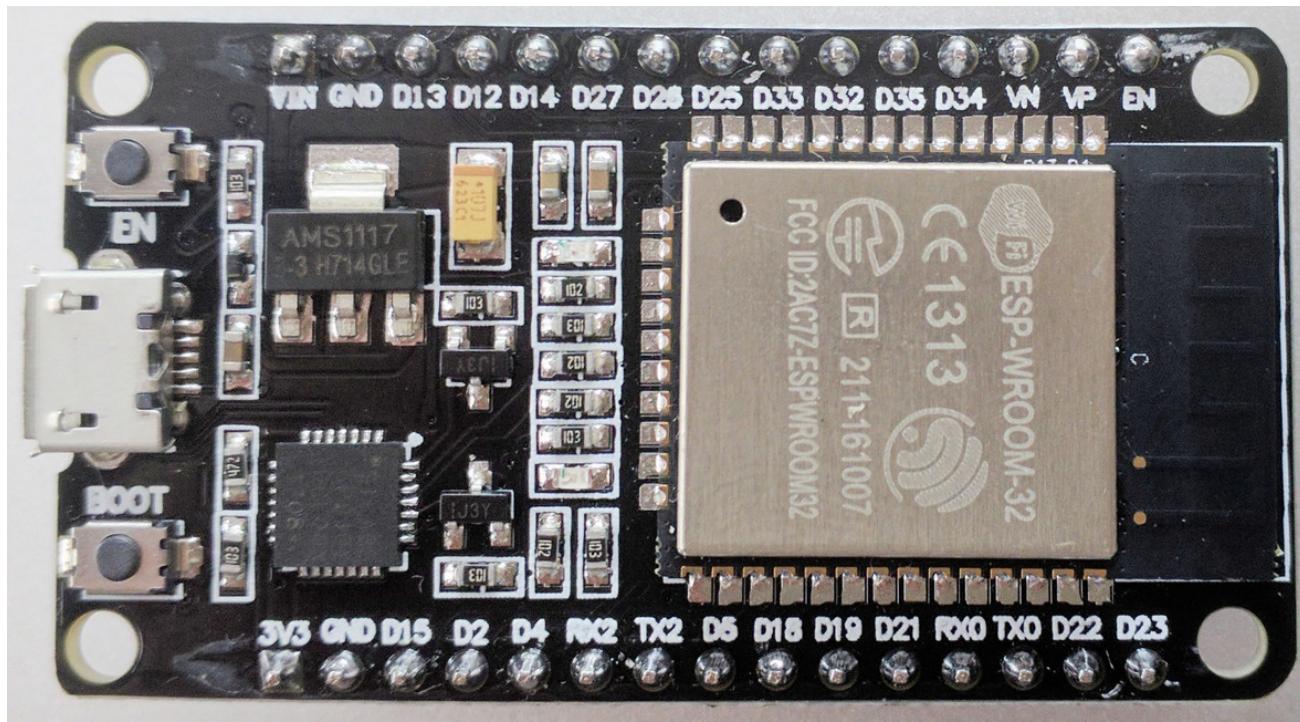
La liaison USB utilise cette liaison



ESP32 + Lua RTOS

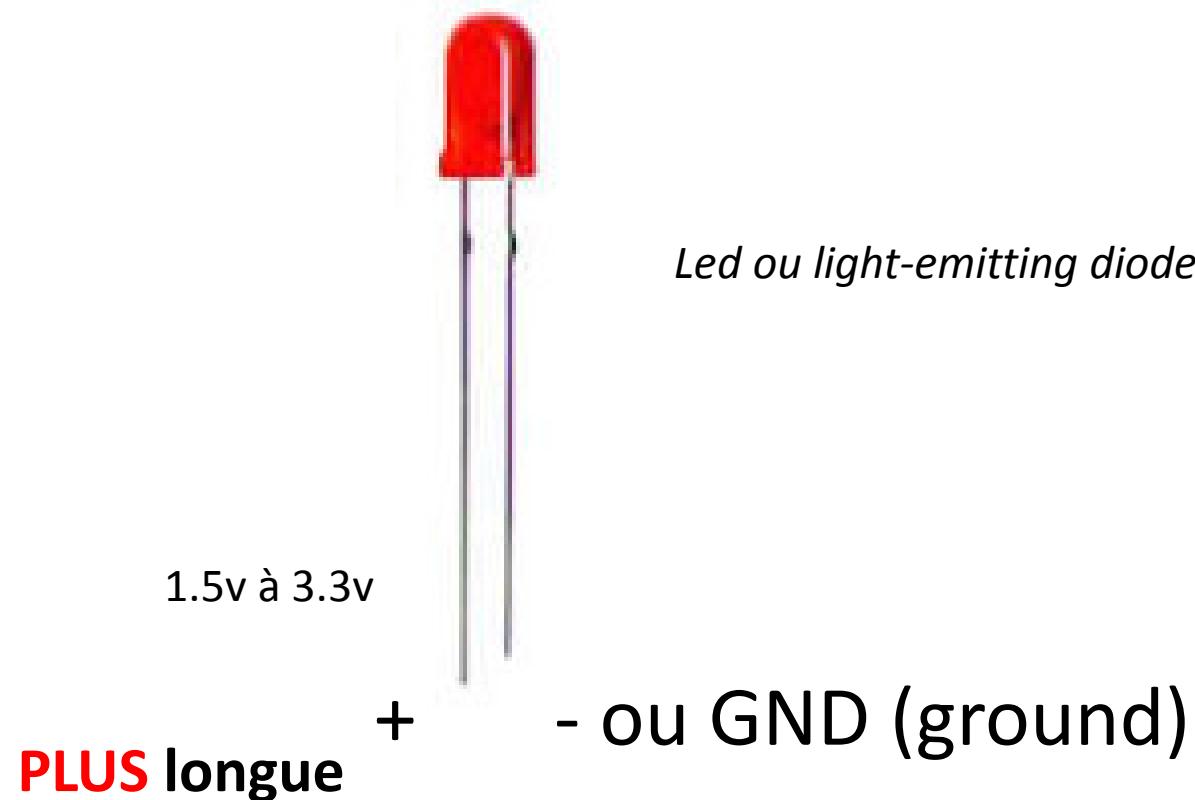
Les pattes notées D2, D4 etc... sont des liaisons électriques que l'esp32 peut utiliser pour communiquer avec l'extérieur.

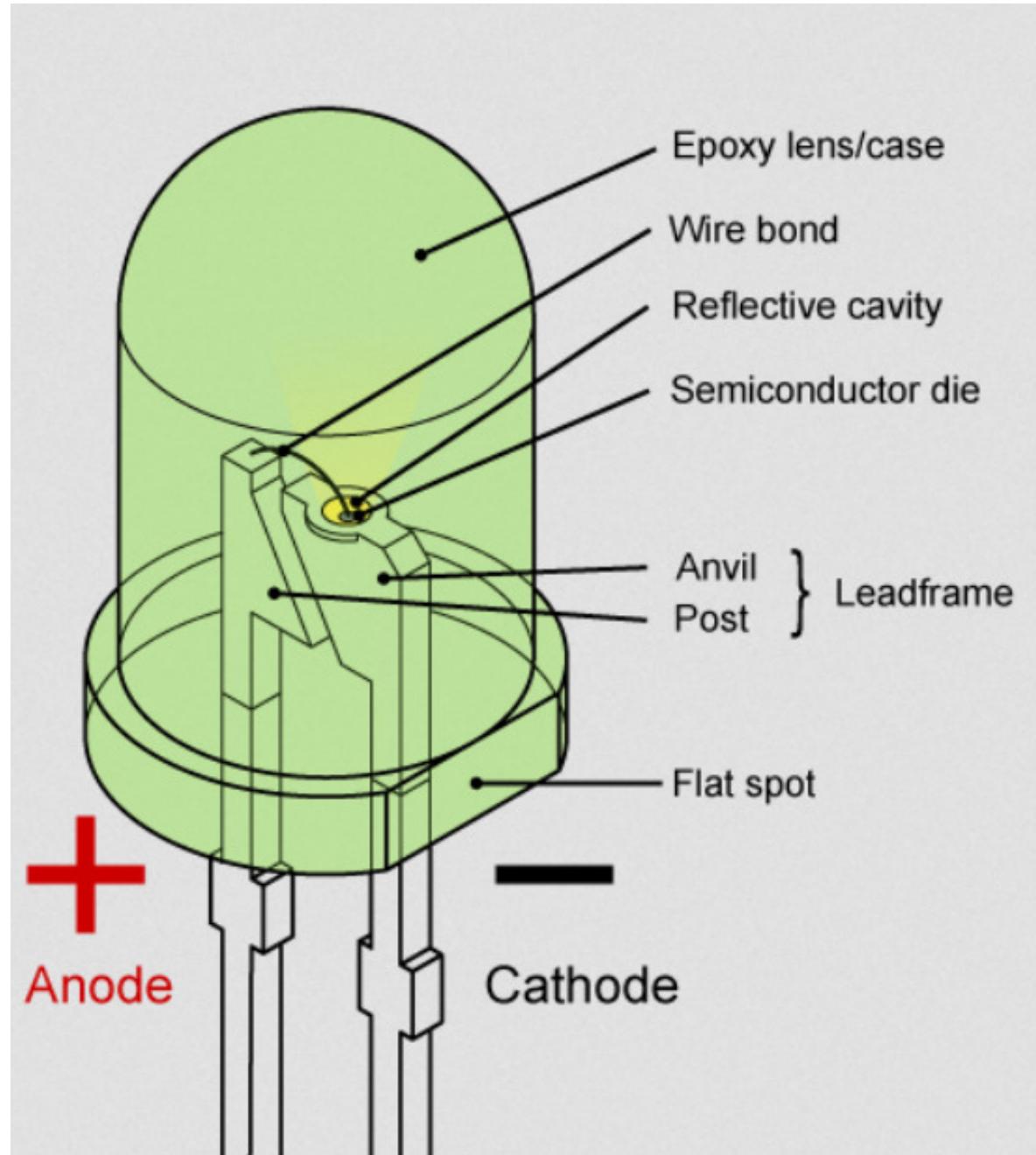
Exemple : la patte D2 et la led bleue qui clignote



ESP32 + Lua RTOS

Vous allez maintenant faire la même chose avec une Led externe





ESP32 + Lua RTOS

Prototypage

Soudure : peu pratique et les composants peuvent être abimés

Plaques (ou breadboard – planches à pain)

pratique

MAIS bazar à gérer

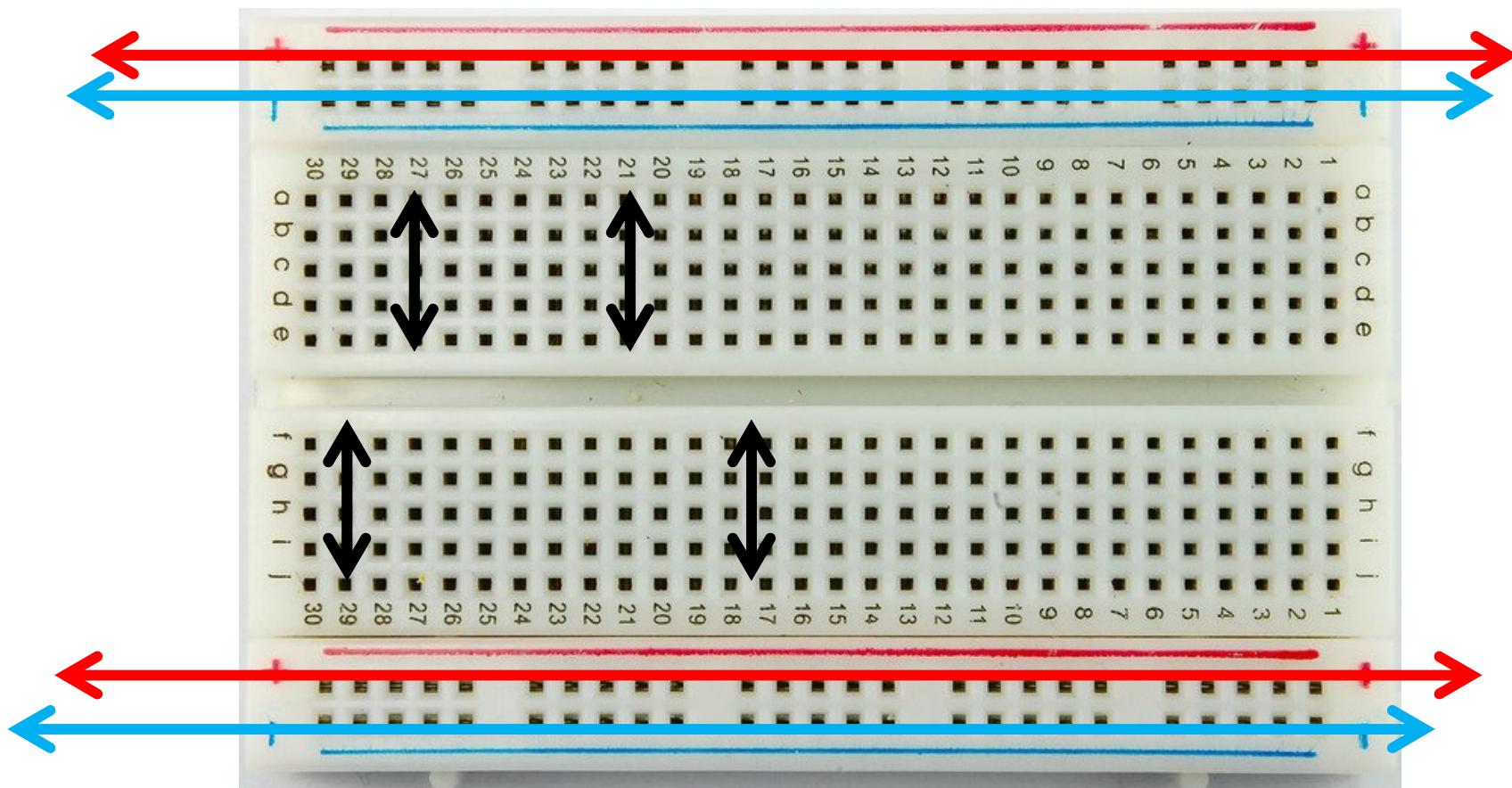
MAIS contacts pas toujours parfait

CONSIGNES :

1. Toujours réaliser les montages débranchés (hors tension)
2. Faire vérifier par un camarade
3. Me demander de vérifier avant de brancher
4. Débrancher la prise au niveau du PC et pas au niveau de l'ESP32

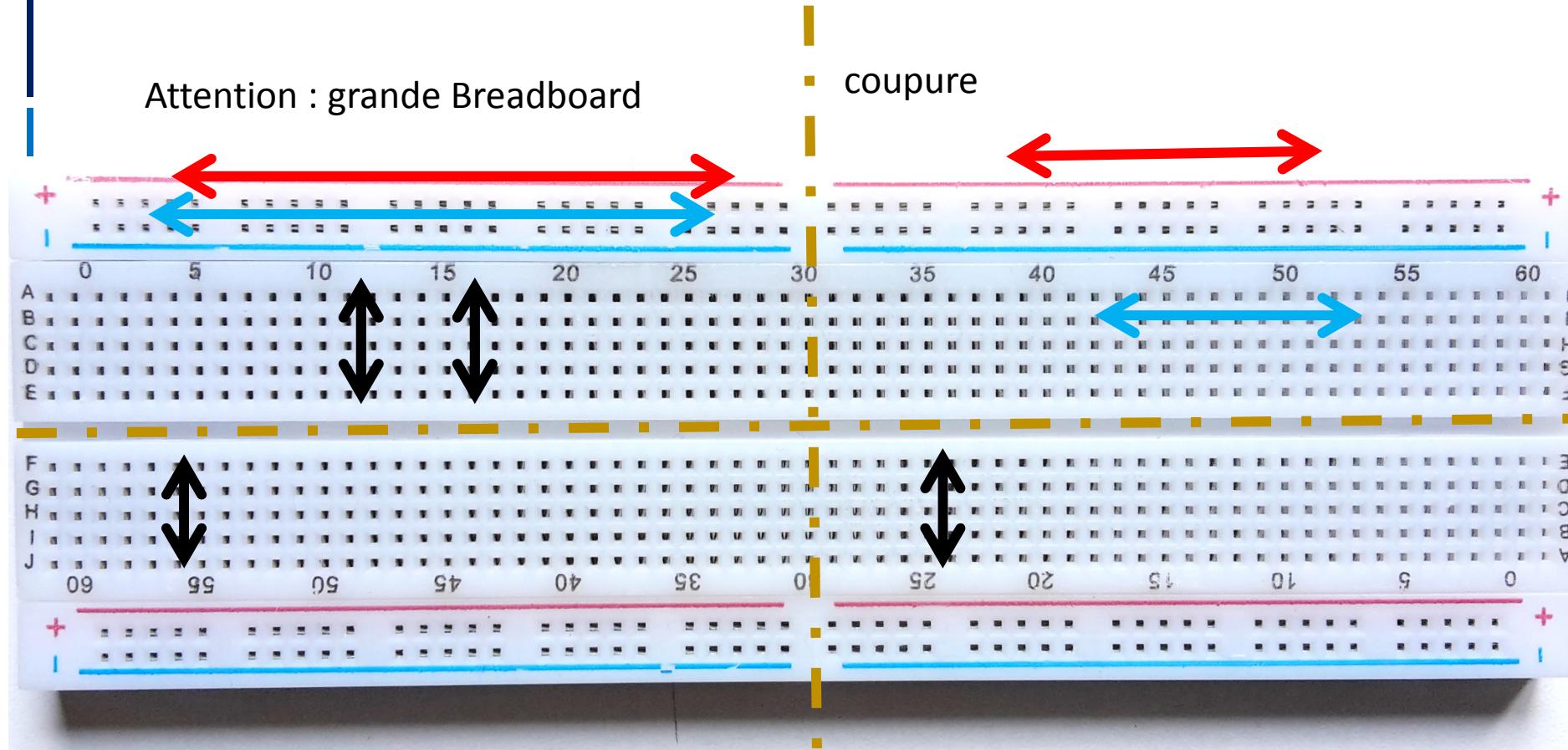
ESP32 + Lua RTOS

Breadboard



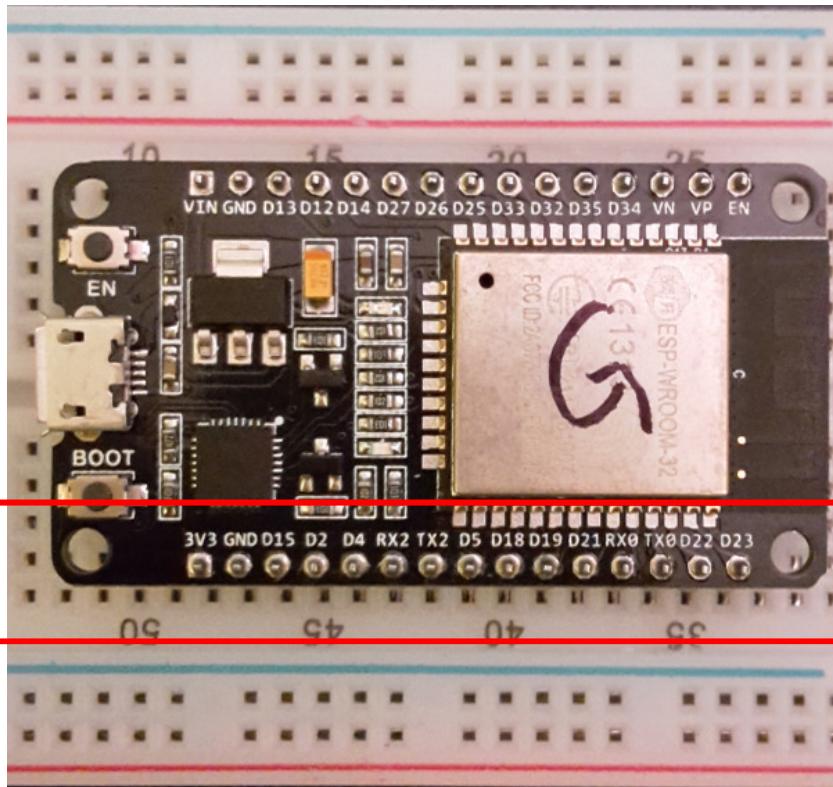
ESP32 + Lua RTOS

Attention : grande Breadboard

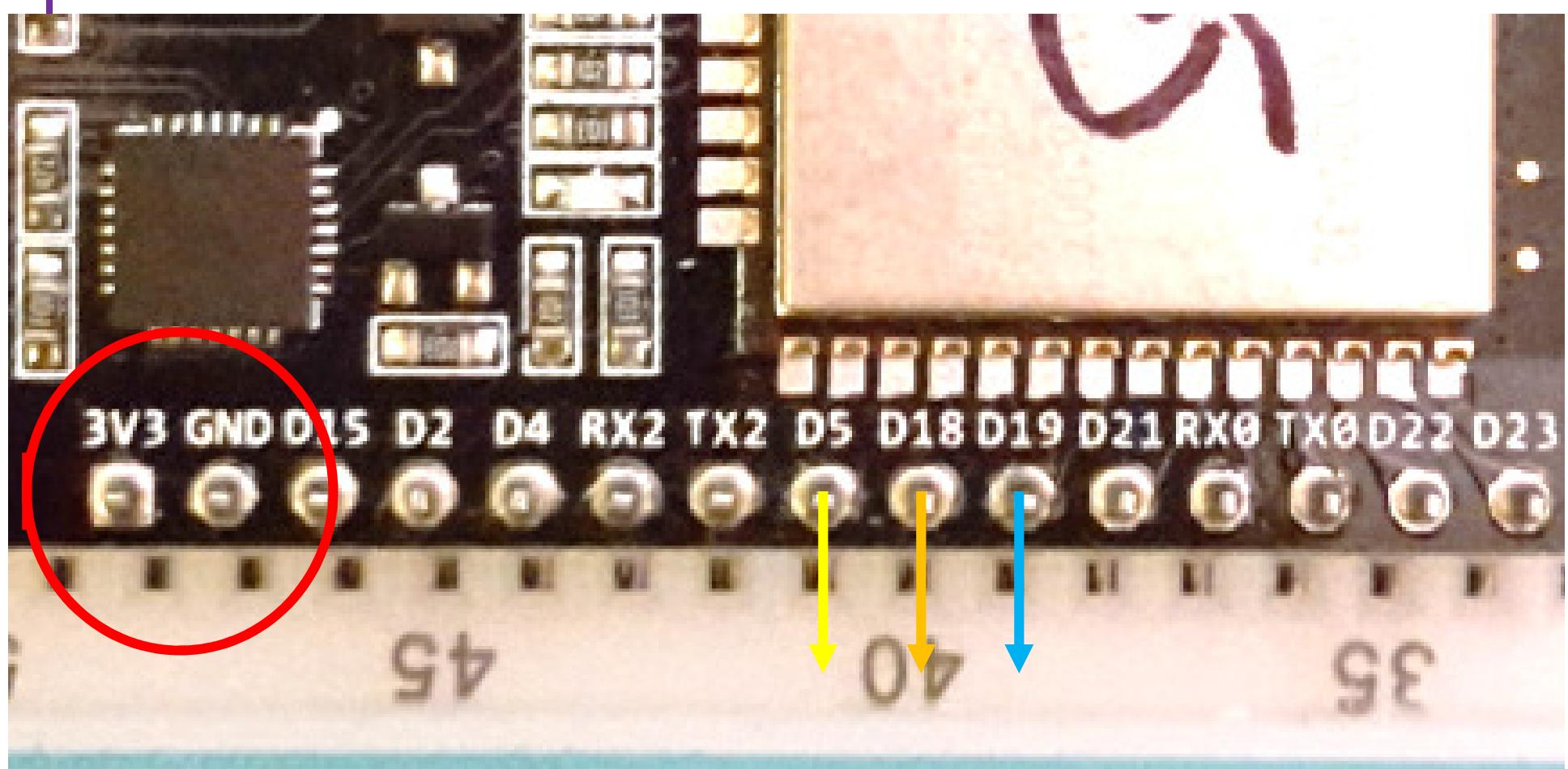


ESP32 + Lua RTOS

Enficher VERTICALEMENT l'ESP32 sur la breadboard ATTENTION AUX PATTES
Laisser une rangée de trous en bas (une seule)



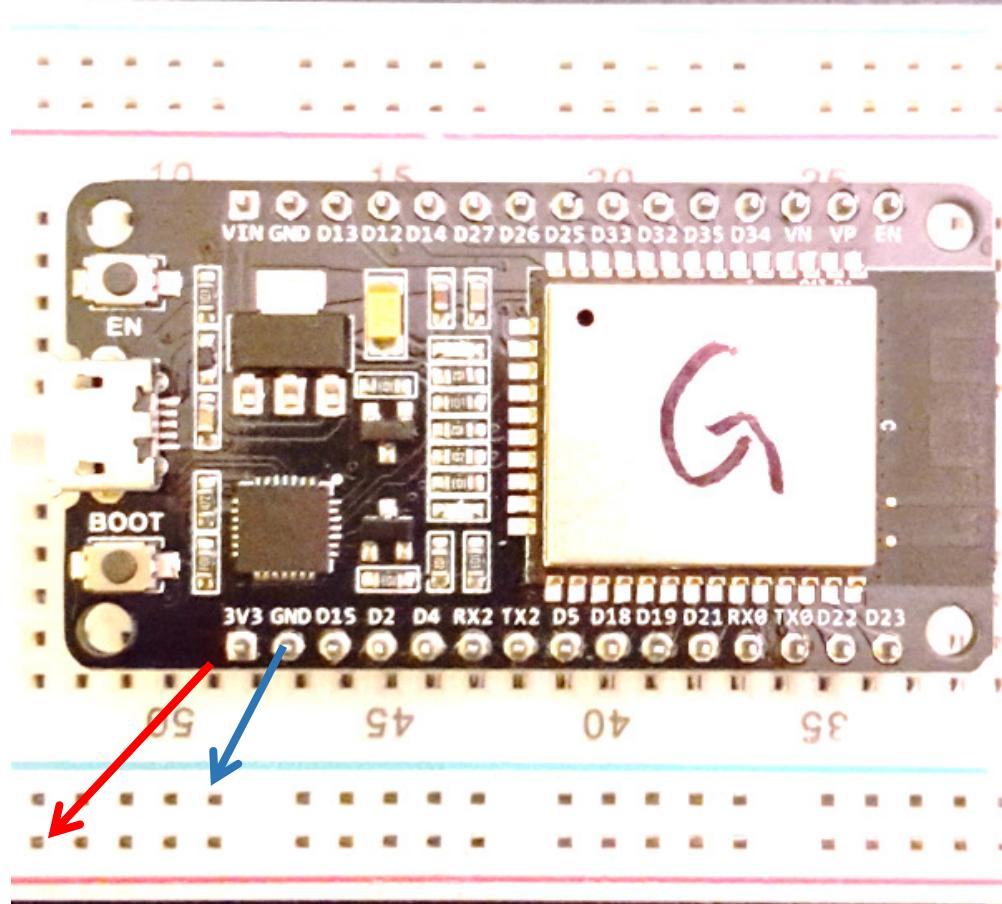
On va récupérer l'alim 3V3
et la masse de l'ESP32 :



Les pattes sont connectées aux contacts en dessous

ESP32 + Lua RTOS

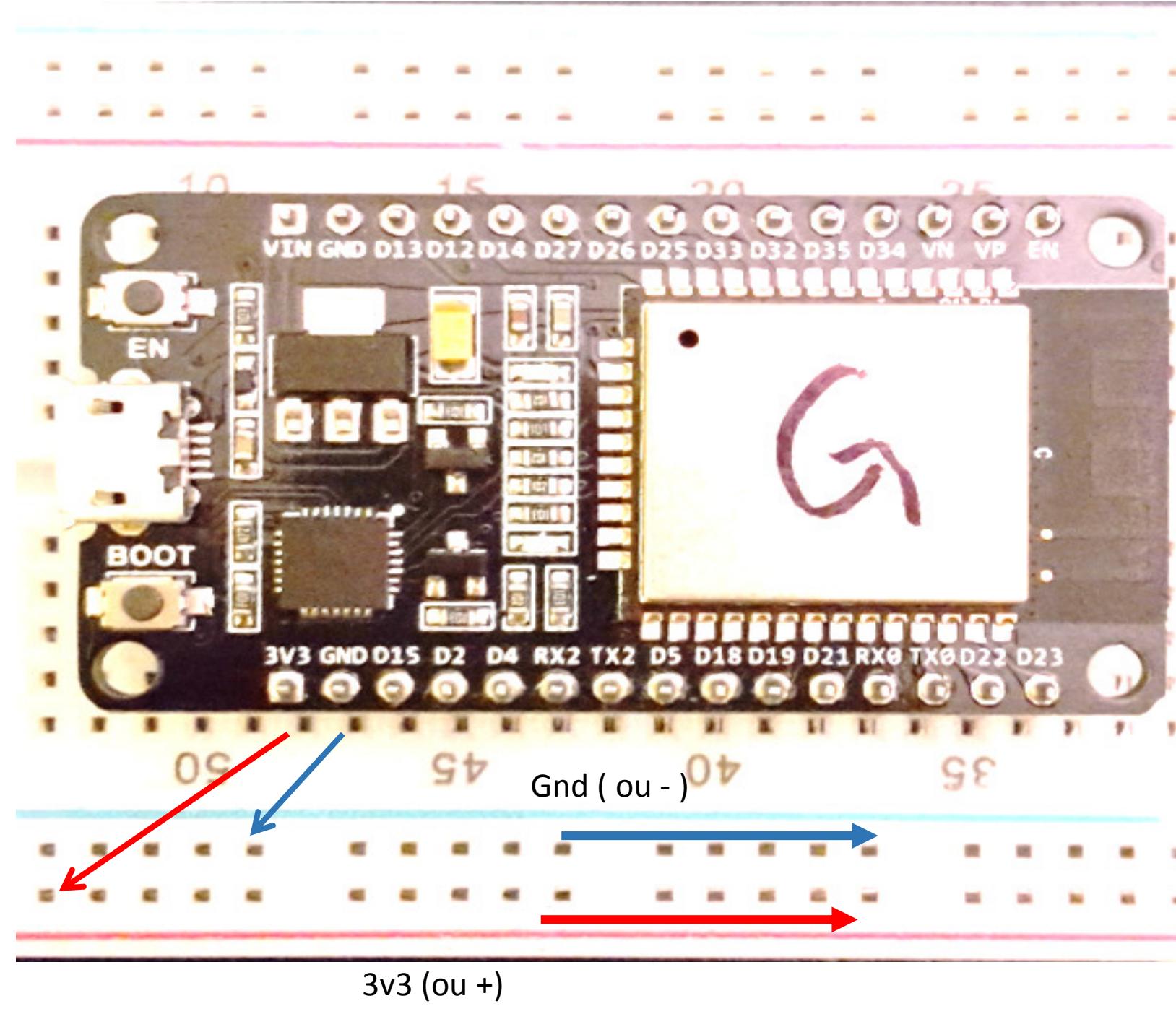
ESP32 Débranché du port USB



Relier le 3v3 à la bande + (rouge)

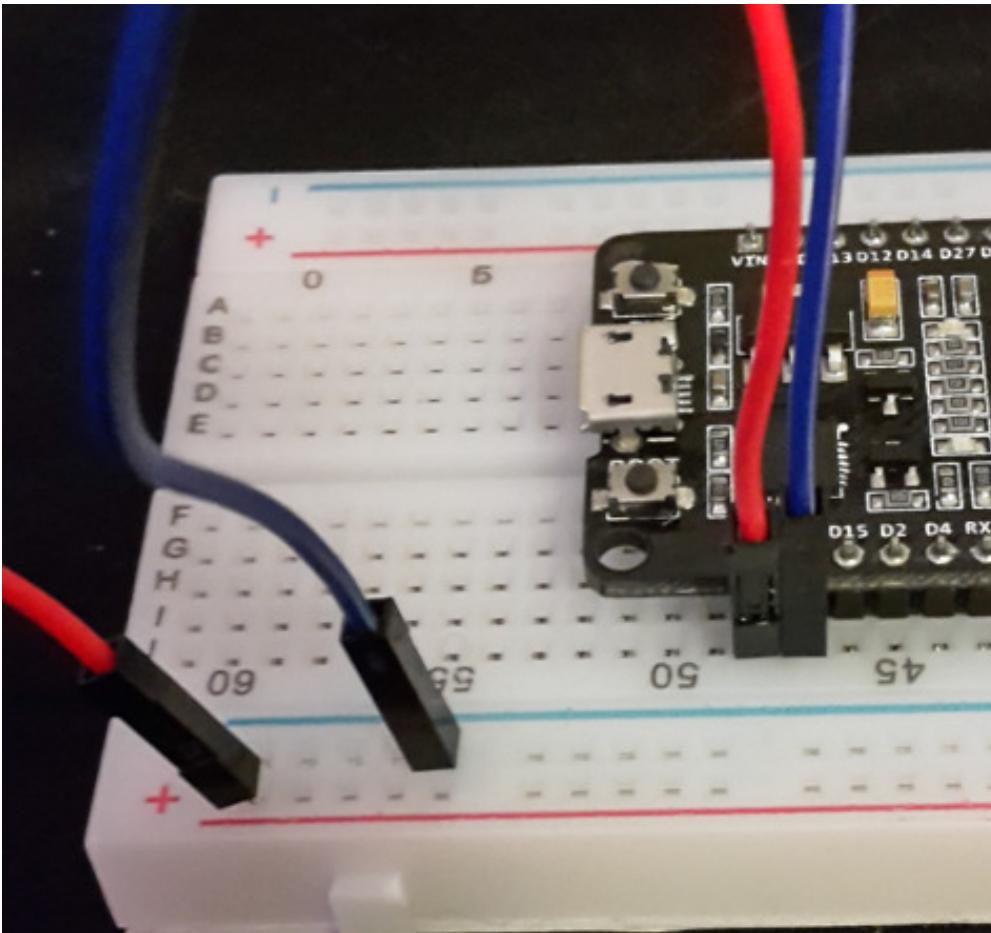
Relier le GND) la bande – (bleu)

Prenez un câble rouge et un bleu



ESP32 + Lua RTOS

ESP32 Débranché du port USB



Faites vérifier par au moins
2 personnes avant de brancher

Règle no.1 : ne vous faites pas
confiance pour les branchements

Règle no.2 : appliquez la règle 1

Règle no.3 : recommencez

ESP32 + Lua RTOS

La led bleu de l'esp32 est allumée quand la patte D2 fournit du +3.3V

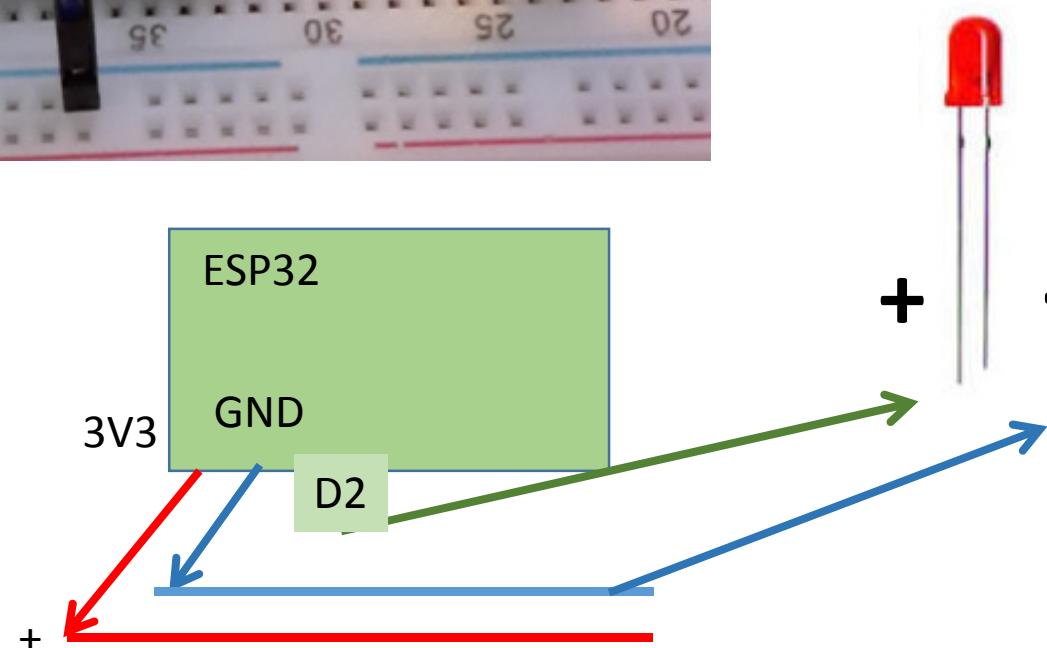
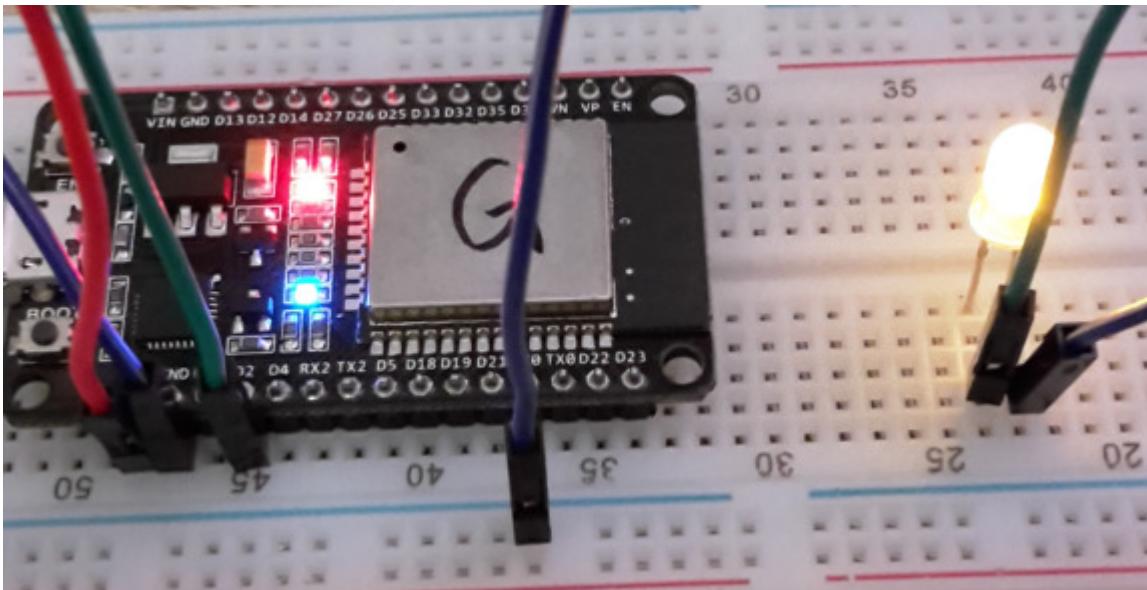
```
pio.pin.sethigh(pio.GPIO2)
```

On utilise la patte D2 à la place du + pour la LED

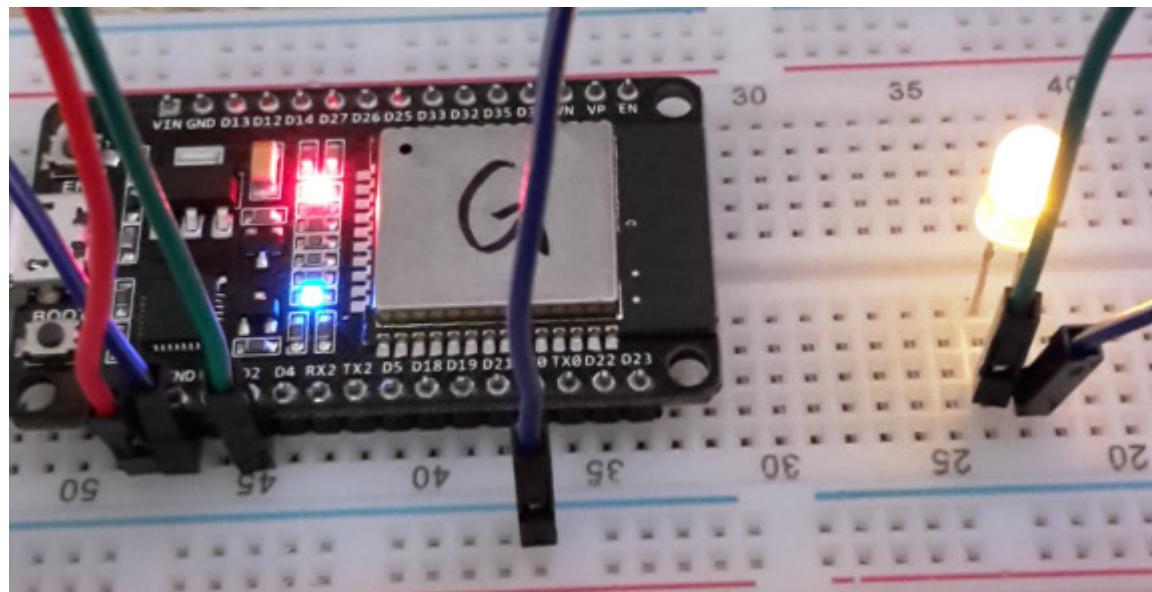
- de la LED : GND (bleu)
- + de la LED : D2

(l'esp32 controle si D2 est à + ou 0)

Vous allez utiliser la breadboard pour connecter l'anode de la led à la patte D2
Et la cathode, évidemment à la bande -



Faire clignoter la led bleue et vérifier que la led rajoutée clignote également



La led doit clignoter
en même temps que
la led bleu de l'esp

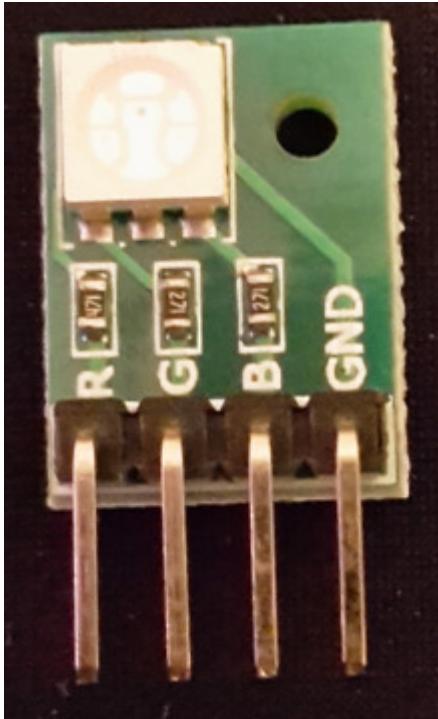
ESP32 + Lua RTOS

Débranchez la led précédente.

Vous allez utiliser 3 leds : rouge, vert et bleu

Elles sont miniaturisées et ont une cathode (-) commune

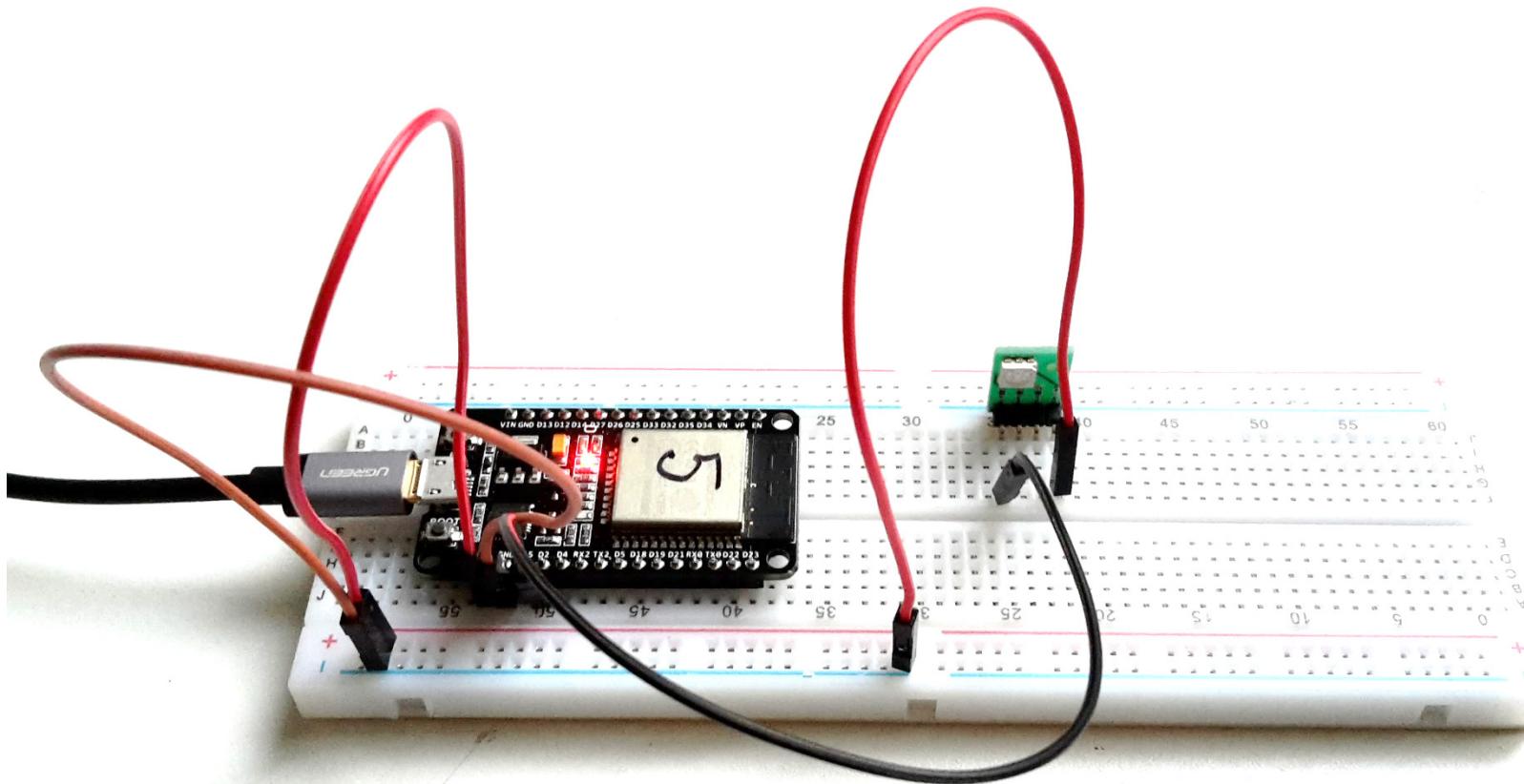
Elles sont protégées par des mini résistances. La masse est commune.



 + + + (-)

Commencez par enficher les pattes dans la breadboard
Puis connectez – en premier
Enfin, branchez GPIO2 / D2 sur **R+**
et vérifiez que cela clignote (rouge)

ESP32 + Lua RTOS



ESP32 + Lua RTOS

On souhaite maintenant libérer D2 et utiliser D18 pour le clignotement. On souhaite faire clignoter de la manière suivante :

```
dofile("blink2.lua")
```

Créez un fichier blink2.lua :

```
pio.pin.setdir(pio.OUTPUT, pio.GPIO18)
```

```
for j=1,10 do
    pio.pin.sethigh(pio.GPIO18)
    tmr.delayms(500)
    pio.pin.setlow(pio.GPIO18)
    tmr.delayms(500)
end
```

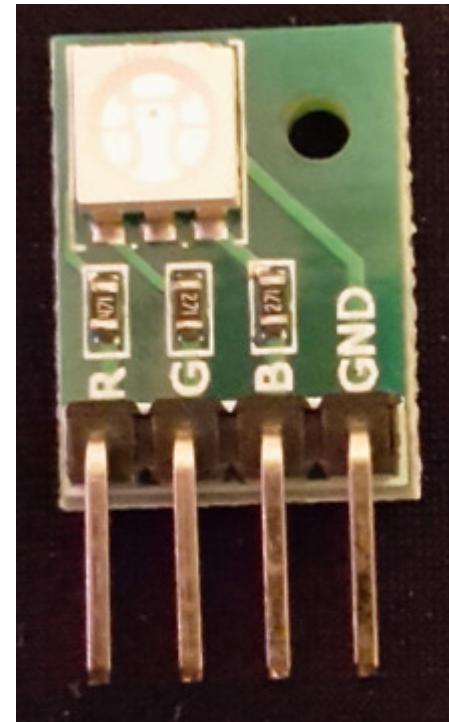
ESP32 + Lua RTOS

Changez le branchement branchez D18 sur G
au lieu de R. Lancez dofile("blink2.lua")

Que remarquez-vous ?

Changez : D18 sur B au lieu de G
Lancez dofile("blink2.lua")

Que remarquez vous ?



d18

d18

d18

ESP32 + Lua RTOS

On souhaite alternativement afficher la couleur rouge (0.5s)
puis verte (0.5s)
puis bleu (0.5s)
puis on recommence... (5 fois)

Rappel : `pio.pin.setdir(pio.OUTPUT, pio.GPIO18)`

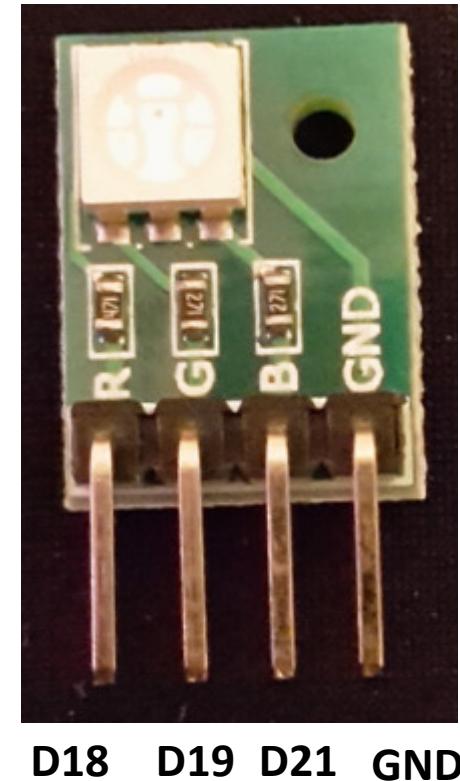
Pour déclarer une patte en écriture / sortie

`pio.pin.sethigh(pio.GPIO18)`

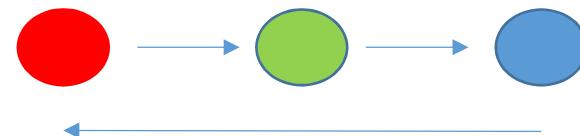
pour forcer la patte en +3.3V

`pio.pin.setlow(pio.GPIO18)`

pour mettre la patte à 0v



D18 D19 D21 GND



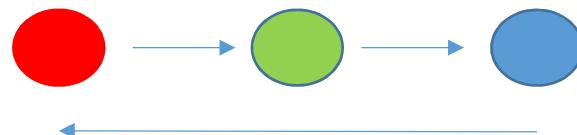
Réalisez `blinkrgb.lua`

ESP32 + Lua RTOS

```
r = pio.GPIO18; g = pio.GPIO19  
b = pio.GPIO21
```

```
pio.pin.setdir(pio.OUTPUT, r)  
pio.pin.setdir(pio.OUTPUT, g)  
pio.pin.setdir(pio.OUTPUT, b)
```

```
for j=1,5 do  
    pio.pin.sethigh(r); tmr.delayms(500); pio.pin.setlow(r)  
    pio.pin.sethigh(g); tmr.delayms(500); pio.pin.setlow(g)  
    pio.pin.sethigh(b); tmr.delayms(500); pio.pin.setlow(b)  
end
```



ESP32 + Lua RTOS

Le dispositif remplace 3 leds qui sont très proches

On peut aussi brancher
3 vraies led et
obtenir le même résultat

Que donne comme couleur

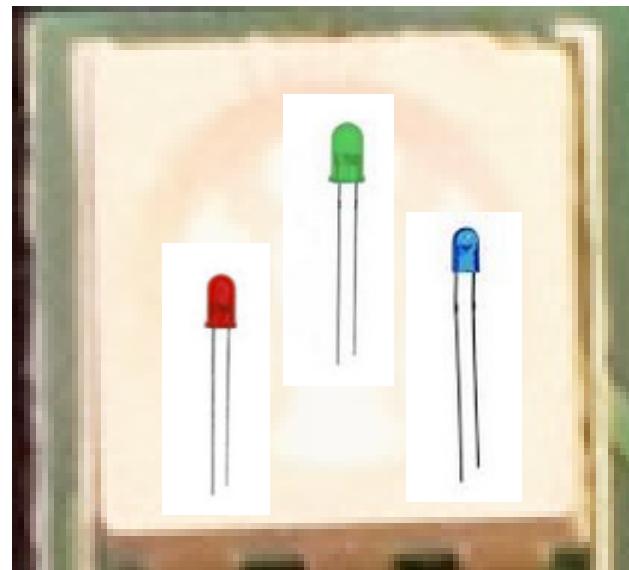
rouge + vert ?

rouge + bleu ?

vert + bleu ?

rouge+vert+bleu ?

Essayez en ligne de commande



ESP32 + Lua RTOS

Le dispositif remplace 3 leds qui sont très proches

On peut aussi brancher
3 vraies led et
obtenir le même résultat

Que donne comme couleur

rouge + vert ?



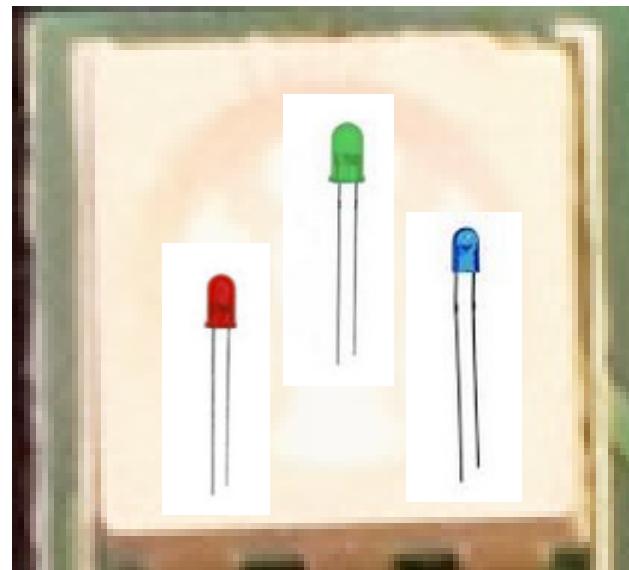
rouge + bleu ?



vert + bleu ?



rouge+vert+bleu ?



PWM

Tension des pattes du microcontrôleur : 0v (GND) et VCC (3.3v)
Sortie binaire (Digitale)

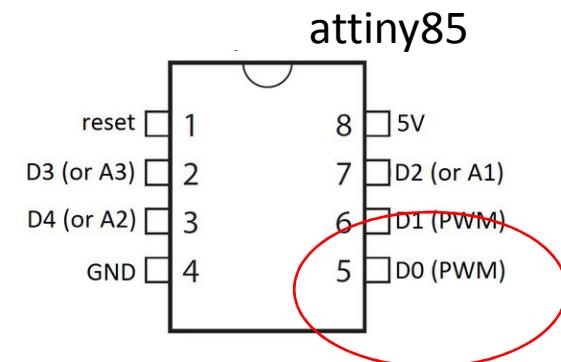
On veut faire varier la tension de sortie entre 0 et 3.3v

Utilisation de transitions rapides 0 – 3.3v

Modulation de largeur d'impulsion (MLI)

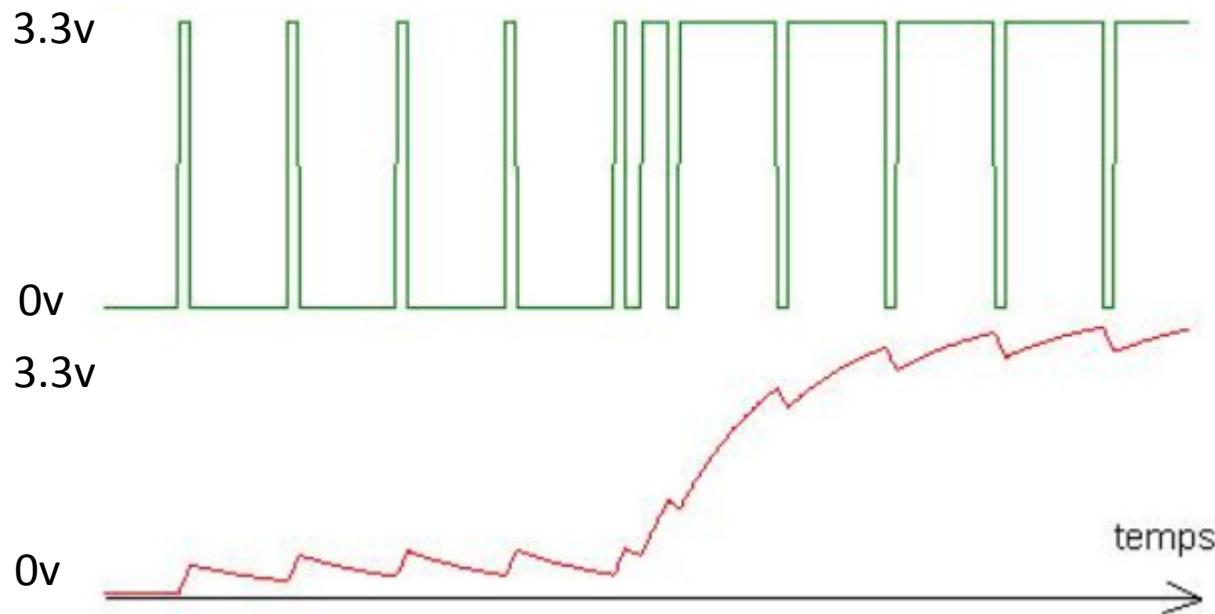
Ou Pulse Width Modulation (**PWM**)

On passe d'états discrets à un état continu : le courant est obtenu par moyenne



PWM

Tension des pattes du microcontrôleur : 0v (GND) et VCC (3.3v)



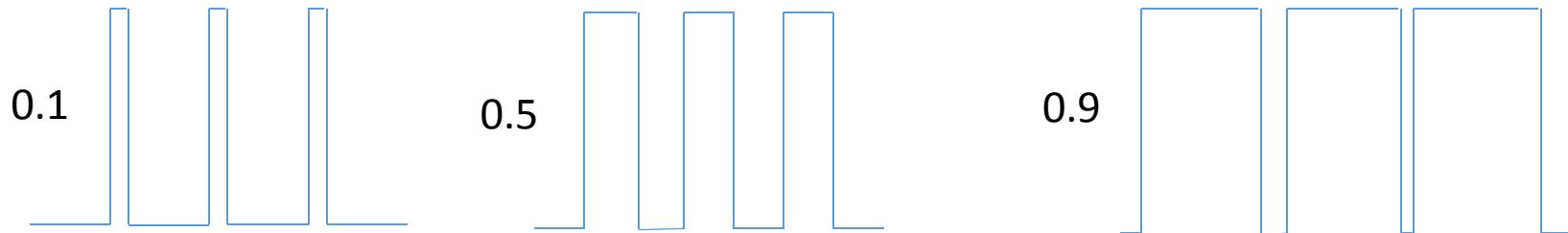
La moyenne est réalisée par le périphérique (exemple led et rémanence)
ou bien directement par le micro contrôleur (patte PWM)

PWM

```
device = pwm.attach(pin, frequency, initial duty)
```

Frequency : fréquence de vibration (en hertz)

Duty : largeur de bande (entre 0 et 1)



```
device:start()
```

```
device:setduty(duty)
```

```
device:setfreq(frequency)
```

```
device:stop()
```

```
device:detach()
```

PWM

Faire clignoter la led rouge (**SANS BOUCLE**) :

Brièvement une fois par seconde
4 fois par seconde

Longtemps une fois par seconde
4 fois par seconde

Faire allumer progressivement la led

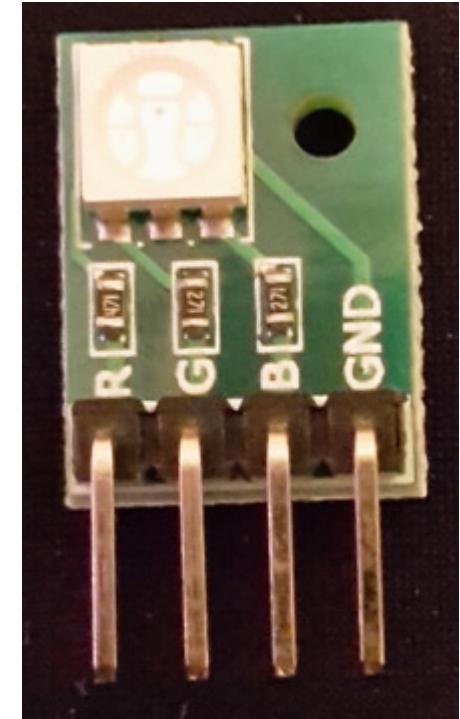
Faire éteindre progressivement la led

PWM

En modulant les rvb, afficher toutes les couleurs saturées

Faire une fonction `rvb(r,v,b)` avec `r,v,b` 0-255 qui allume les leds avec l'intensité souhaitée

Utiliser cette fonction avec 6 boucles



PWM

Ou 256
(ici moins lumineux)
(param supp)

```
r = pwm.attach(pio.GPIO18, 5000, 0.5)
v = pwm.attach(pio.GPIO19, 5000, 0.5)
b = pwm.attach(pio.GPIO21, 5000, 0.5)

rvb= function(rr,vv,bb)
    r:setduty(rr/512)
    v:setduty(vv/512)
    b:setduty(bb/512)
end

r:start(); v:start(); b:start();

rvb(0,0,0)

for i=1,255,3 do rvb(i,255,0); tmr.delayms(20) end
for i=255,1,-3 do rvb(255,i,0); tmr.delayms(20) end
for i=1,255 do rvb(255,0, i); tmr.delayms(20) end
for i=255, 1, -3 do rvb(i,0,255); tmr.delayms(20) end
for i=1,255 do rvb(0,i,255); tmr.delayms(20) end
for i=255, 1,-3 do rvb(0, 255, i); tmr.delayms(20) end

r:stop(); r:detach()
v:stop(); v:detach()
b:stop(); b:detach()
```



ESP32 + Lua RTOS

On peut modifier l'intensité lumineuse de chaque led

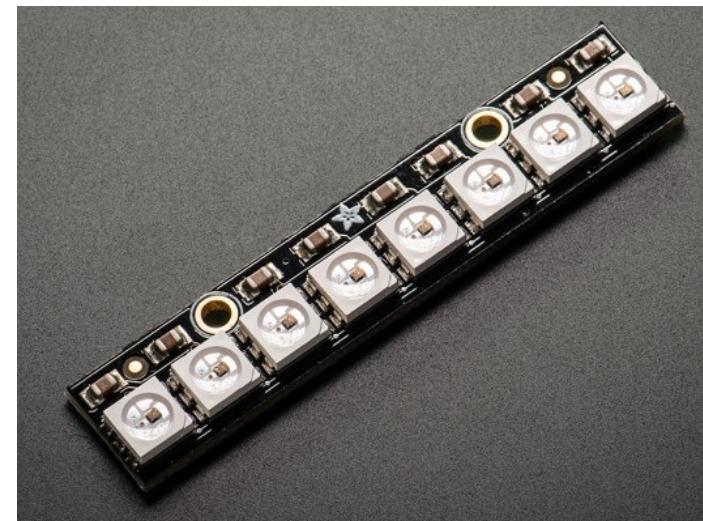
et régler la proportion de rouge, vert et bleu et

créer n'importe quelle couleur en changeant la tension sur
chaque patte D18, D19, D21.

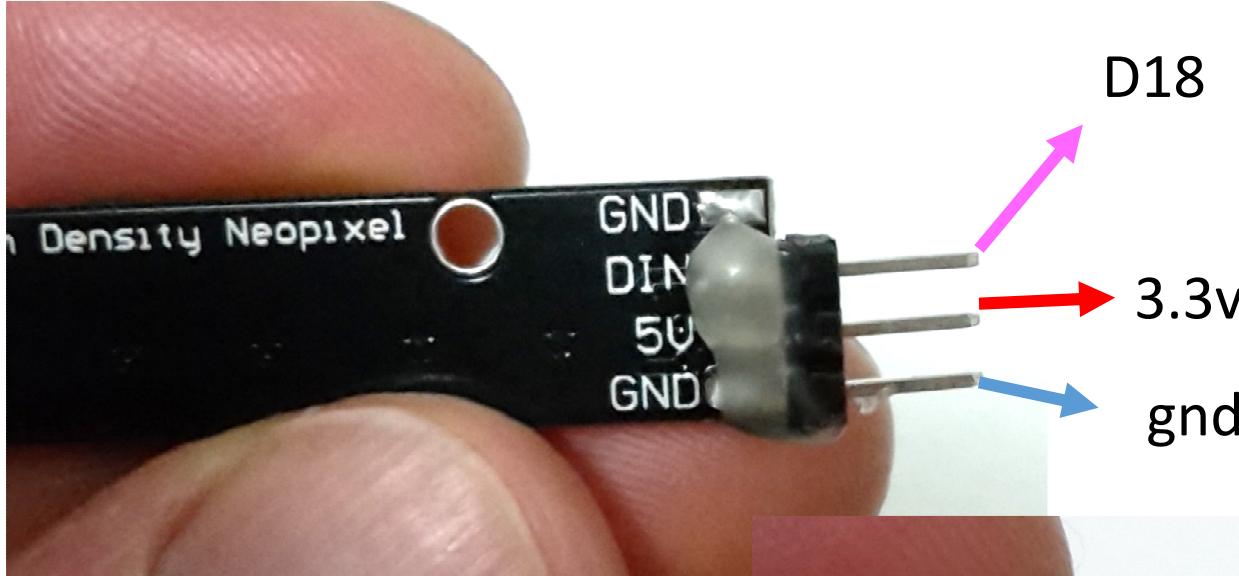
Ou bien utiliser des neopixels :
ce sont des leds spéciales.

WS2812

Specifs technique à étudier

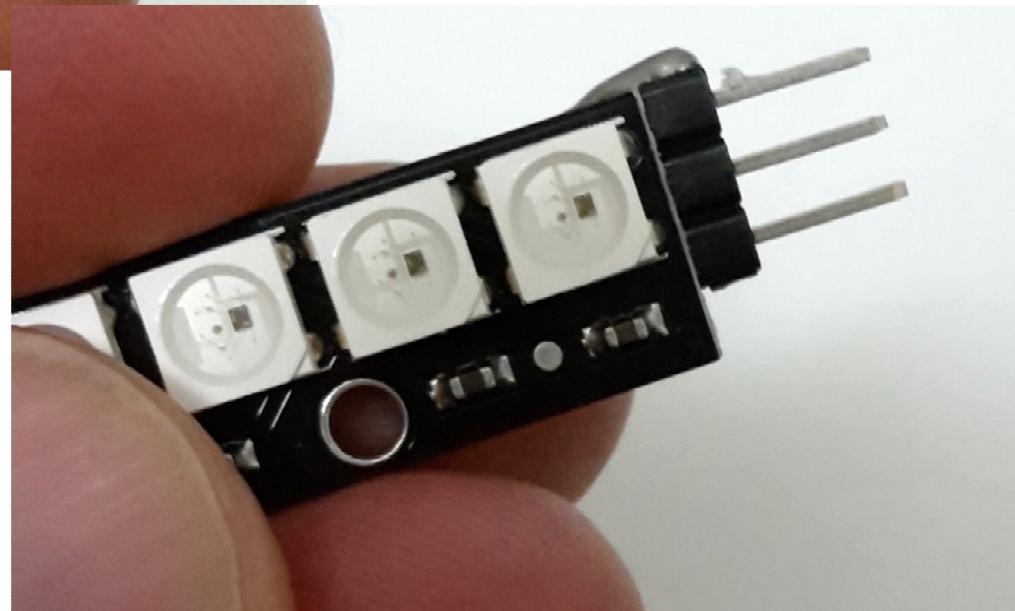


ESP32 + Lua RTOS



Plusieurs neopixel

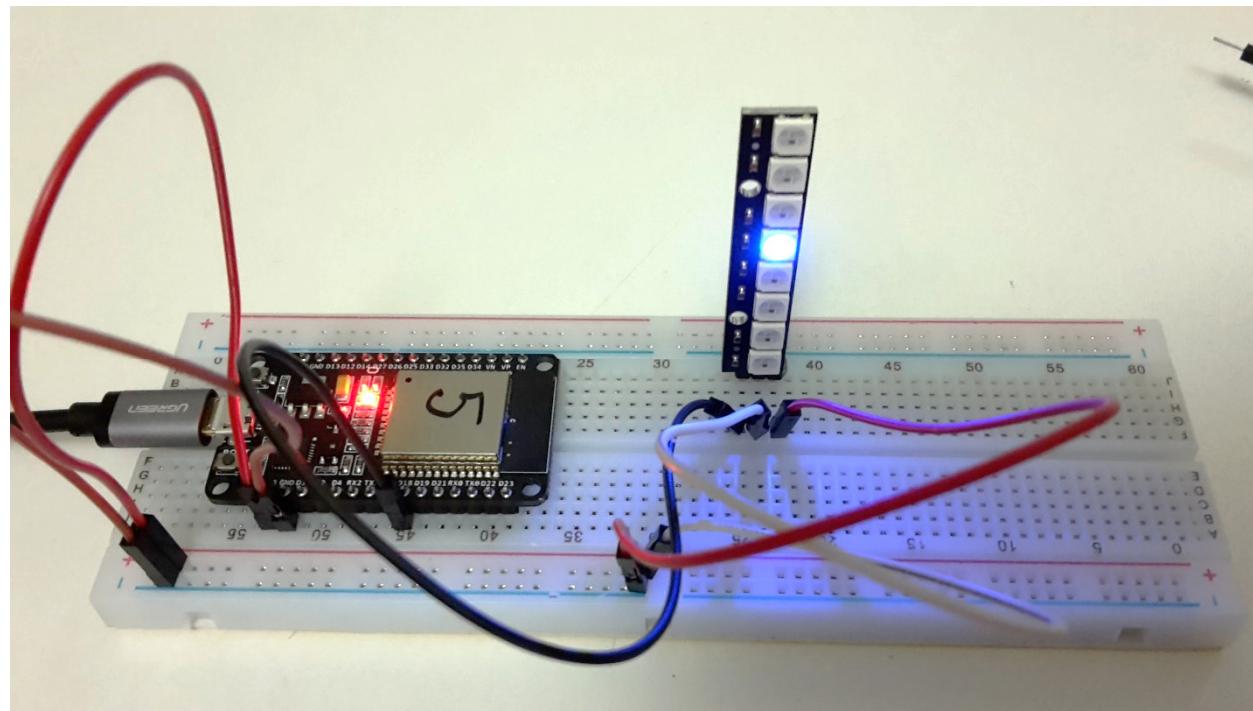
On peut contrôler celui qu'on souhaite avec la couleur qu'on veut



ESP32 + Lua RTOS

Dans la console (en ligne de commande)

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8)  
neo:setPixel(4, 0,0,20)  
neo:update() ;
```



```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8)
```

attention

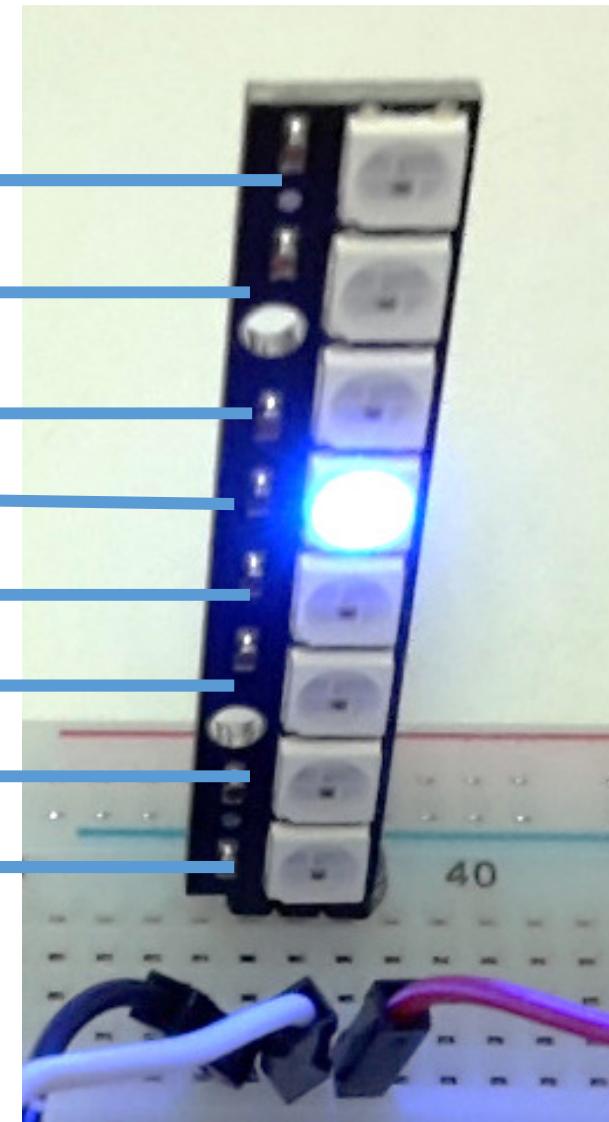
```
neo:setPixel(4, 0,0,20)
```

Rouge
0-255

Vert
0-255

7
6
5
4
3
2
1
0

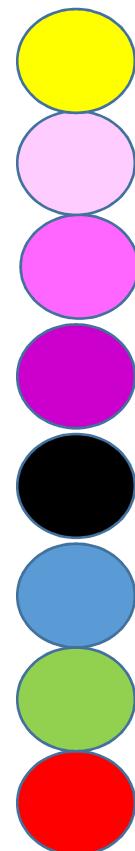
```
neo:update()
```





ESP32 + Lua RTOS

Exercice :



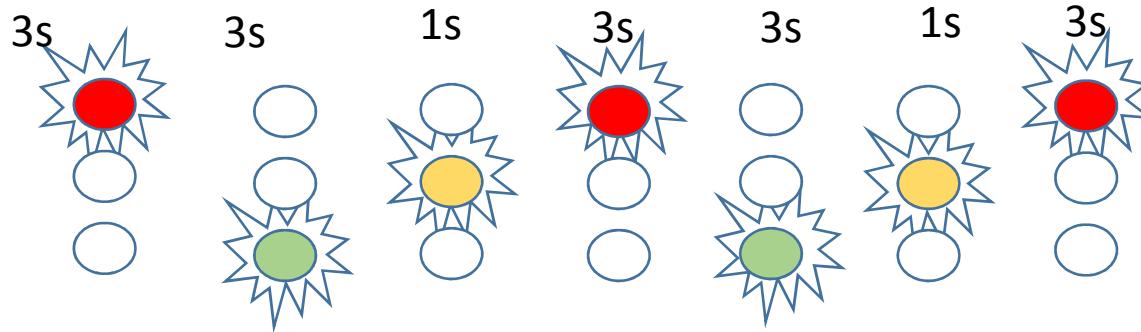
R

V

B

ESP32 + Lua RTOS

On veut réaliser le microcontrôleur d'un feu de circulation :



Comme un feu ne s'arrête pas, boucle infinie

while true do

.....

end

Rappel :

`neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8)`

`neo:setPixel(4, 0,0,20)`

`neo:update()` ;

reboot

Créer un fichier feux.lua et lancer par dofile("feux.lua")

ESP32 + Lua RTOS

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8)

for j=0,7 do neo:setPixel(j,0,0,0) end
neo:update()

while true do
    neo:setPixel(7, 20,0,0); neo:update()
    tmr.delay(3); neo:setPixel(7, 0,0,0)
    neo:setPixel(5, 0,20,0); neo:update()
    tmr.delay(3); neo:setPixel(5, 0,0,0)
    neo:setPixel(6, 60,20,0); neo:update()
    tmr.delay(1); neo:setPixel(6, 0,0,0)
end
```

Parfait exemple d'objet connecté : changement délais ou feux oranges ou feux rouges !

- `neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8)`
- `for j=0,7 do neo:setPixel(j,0,0,0) end`
- `neo:update()`
- `while true do`
- `neo:setPixel(7, 20,0,0); neo:update()`
- `tmr.delay(3); neo:setPixel(7, 0,0,0)`
- `neo:setPixel(5, 0,20,0); neo:update()`
- `tmr.delay(3); neo:setPixel(5, 0,0,0)`
- `neo:setPixel(6, 60,20,0); neo:update()`
- `tmr.delay(1); neo:setPixel(6, 0,0,0)`
- `end`

ESP32 + Lua RTOS

Oui mais boucle infinie : le microcontrôleur ne sert qu'à une tâche !

Comment transformer le code pour faire en sorte de libérer le microcontrôleur, qui peut alors contrôler le feu et réaliser d'autres action en même temps ?

(mesure du traffic, émission des informations par internet, signal sonore, gestion des piétons, surveillance caméra etc.)

THREAD !

Rappel : t = thread.start(gestion_feux)

feux_th.lua

```
function gestion_feux()
    neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18, 8)

    for j=0,7 do neo:setPixel(j,0,0,0) end
    neo:update()

    while true do
        neo:setPixel(7, 20,0,0); neo:update()
        tmr.delay(3); neo:setPixel(7, 0,0,0)
        neo:setPixel(5, 0,20,0); neo:update()
        tmr.delay(3); neo:setPixel(5, 0,0,0)
        neo:setPixel(6, 60,20,0); neo:update()
        tmr.delay(1); neo:setPixel(6, 0,0,0)
    end

end

feux_thread = thread.start(gestion_feux)
```

- **function gestion_feux()**
- **neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8)**
- **for j=0,7 do neo:setPixel(j,0,0,0) end**
- **neo:update()**
- **while true do**
- **neo:setPixel(7, 20,0,0); neo:update()**
- **tmr.delay(3); neo:setPixel(7, 0,0,0)**
- **neo:setPixel(5, 0,20,0); neo:update()**
- **tmr.delay(3); neo:setPixel(5, 0,0,0)**
- **neo:setPixel(6, 60,20,0); neo:update()**
- **tmr.delay(1); neo:setPixel(6, 0,0,0)**
- **end**
- **end**
- **feux_thread = thread.start(gestion_feux)**

ESP32 + Lua RTOS

```
dofile("feux_th.lua")
```

```
/ > thread.list()
-----
| THID | TYPE | NAME | STATUS | CORE | PRI0 | SIZE | STACK FREE | USED |
-----  
1073602752 | lua | lua_main | run | 0 | 20 | 10240 | 6164 | 4076 ( 39% )  
1073586440 | lua | lua_thread | run | 1 | 20 | 8192 | 6992 | 1200 ( 14% )  
/ > |
```

Arrêt par reboot ou bien `thread.stop(feux_thread)`

```
/ > thread.list()
-----
| THID | TYPE | NAME | STATUS | CORE | PRI0 | SIZE | STACK FREE | USED |
-----  
1073602752 | lua | lua_main | run | 0 | 20 | 10240 | 6164 | 4076 ( 39% )  
1073586440 | lua | lua_thread | run | 1 | 20 | 8192 | 6992 | 1200 ( 14% )  
/ > thread.stop(feux_thread)  
/ > thread.list()
-----
| THID | TYPE | NAME | STATUS | CORE | PRI0 | SIZE | STACK FREE | USED |
-----  
1073602752 | lua | lua_main | run | 0 | 20 | 10240 | 6164 | 4076 ( 39% )  
/ > |
```

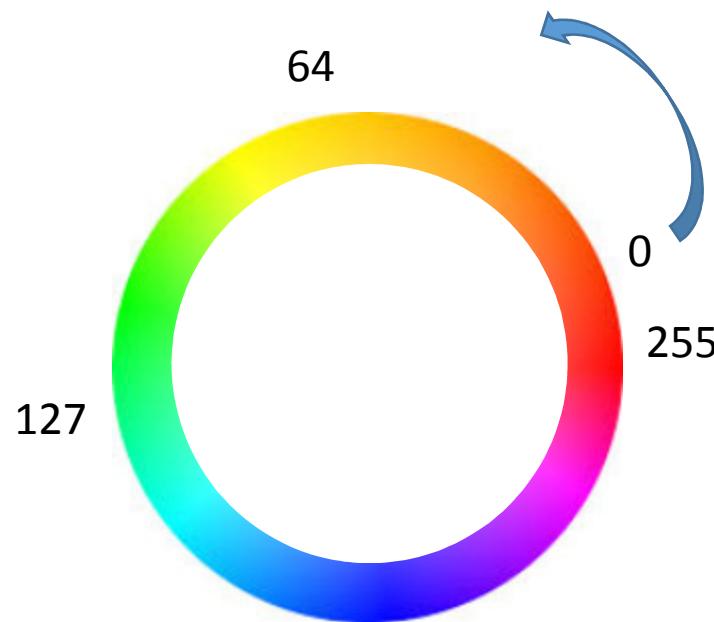
ESP32 + Lua RTOS

dans console.lua

```
function wheelRGB(pos)
    pos = 255 - pos
    if (pos < 85) then
        return 255 - pos * 3, 0, pos * 3
    elseif (pos < 170) then
        pos = pos - 85
        return 0, pos * 3, 255 - pos * 3
    else
        pos = pos - 170
        return pos * 3, 255 - pos * 3, 0
    end
End
```

Conversion 0 – 255 -> r, v, b

Roue des couleurs



Utilisation :

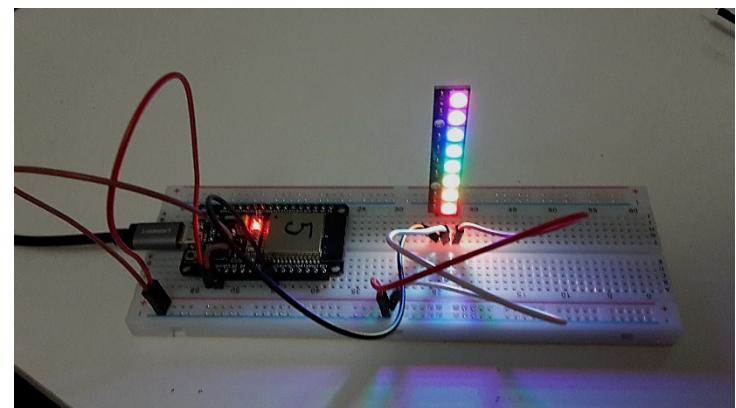
```
r,v,b = wheelRGB(45)
```

ESP32 + Lua RTOS

Créez un fichier arc.lua - lancez par dofile("arc.lua") :

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18, 8)

for j=0,7 do
    r,v,b = wheelRGB((255*j)//8)
    neo:setPixel(j, r//10,v//10,b//10)
end
neo:update()
```



ESP32 + Lua RTOS

Lancez le script `arc_move.lua`

`thread.list()`
`thread.suspend()`
`thread.resume()`
`thread.stop()`

Est-ce un thread ?

Regardez dans le source avec l'éditeur

Maintenant, mettez le script en **PAUSE sans le stopper !**

Remettez-le en marche

Mettez-le en pause exactement 5s

Stoppez le définitivement

```
    thread.start(function()
        neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18, 8)

        pixel = 0
        direction = 0

        while true do
            r, v, b = wheelRGB((255*pixel)//8)
            neo:setPixel(pixel, r, v, b)
            neo:update()
            tmr.delayms(100)
            neo:setPixel(pixel, r//20, v//20, b//20)
            if (direction == 0) then
                if (pixel == 7) then
                    direction = 1; pixel = 6
                else
                    pixel = pixel + 1
                end
            else
                if (pixel == 0) then
                    direction = 0; pixel = 1
                else
                    pixel = pixel - 1
                end
            end
        end
    end)
end)
```

ESP32 + Lua RTOS

math.random(a,b) permet de générer un nombre aléatoire entre a et b

Essayez dans la console print(math.random(0,255))

Réalisez un script « fete.lua » qui fonctionne dans un thread infini et qui change aléatoirement les lumières et les couleurs.

Utilisez des valeurs entre 0 et 20 si vous souhaitez quelque chose de moins lumineux.

Entre deux changements de couleurs, mettez une pause tmr.delayms(



ESP32 + Lua RTOS

1. Réalisez SANS thread :

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18, 8)
```

```
while true do
    r = math.random(0,20)
    v = math.random(0,20)
    b = math.random(0,20)
    neo:setPixel(math.random(0,8), r,v,b)
    neo:update()
    tmr.delayms(20)
end
```

ESP32 + Lua RTOS

2. Rajoutez le thread :

```
function fete()

    neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18, 8)

    while true do
        r = math.random(0,20)
        v = math.random(0,20)
        b = math.random(0,20)
        neo:setPixel(math.random(0,8), r,v,b)
        neo:update()
        tmr.delayms(20)
    end

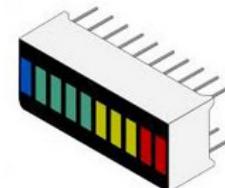
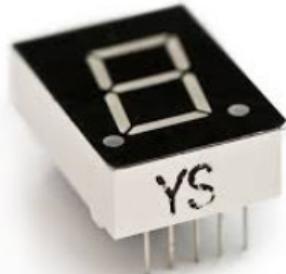
end

fete_thread = thread.start(fete)
```

ESP32 + Lua RTOS

Neopixel, led etc.. : dispositif de communication

Le microcontrôleur peut les utiliser pour transmettre de l'information visuelle



ESP32 + Lua RTOS

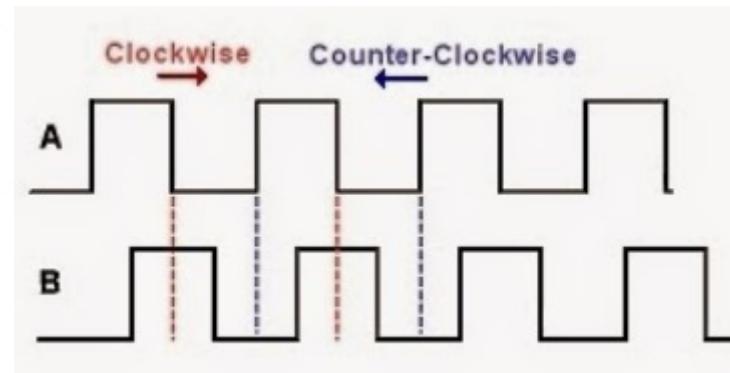
Jusqu'à présent le microcontrôleur -> sortie
Maintenant <- entrée

Utilisateur d'un encodeur rotatif



Tourne (sans fin)
'clicks'

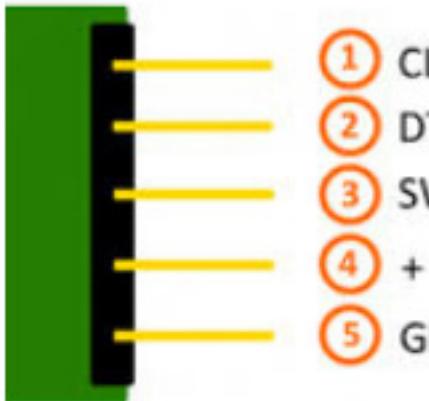
Chaque click met en contact deux pattes
Compter le nombre de click : vitesse
Dans un sens ou dans l'autre



ESP32 + Lua RTOS

Jusqu'à présent le microcontrôleur -> sortie
Maintenant <- entrée

Utilisateur d'un encodeur rotatif



1 CLK Clock D19

2 DT Data D21

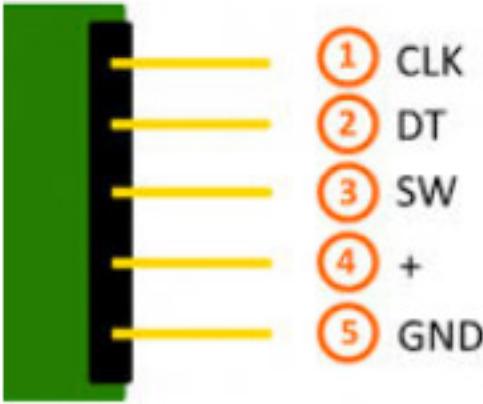
3 SW Bouton poussoir (switch) D5

4 +

5 GND



ESP32 + Lua RTOS

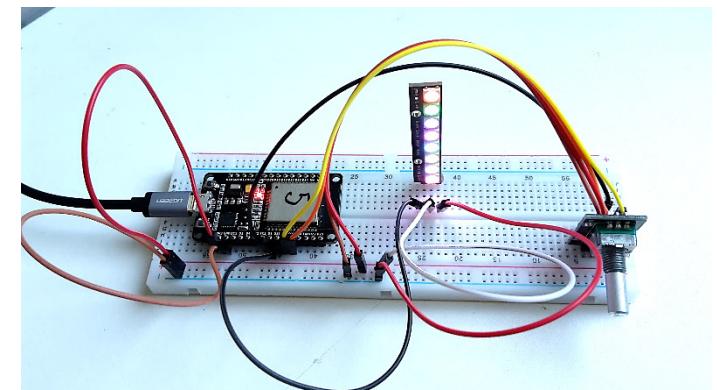


1 CLK Clock D19
2 DT Data D21
3 SW Bouton poussoir (switch) D5
4 +
5 GND



Clock A Data B Bouton poussoir (switch)

enc = encoder.attach(pio.GPIO19, pio.GPIO21, pio.GPIO5) Ou GPIO4
p, s = enc:read() – position et switch (system0)



Normalement, il faudrait tester et lire l'encodeur sans arrêt :

```
enc = encoder.attach(pio.GPIO19, pio.GPIO21, pio.GPIO5)
```

```
while true do
```

```
    p, s = enc:read()
```

```
    print(p, s)
```

```
end
```

Ou GPIO4
(system0)

**Mauvaise idée : il faudrait détecter les changements sinon
on lit plusieurs fois les mêmes valeurs + on occupe le processeur
pour rien ! (consommation etc...)**

**LUA est proche de Javascript : langage fonctionnel asynchrone
(voir node.js)**

Callback = fonction à appeler quand il se passe qq chose

ESP32 + Lua RTOS

reboot

```
function lecture(dir, counter, button)
    print("direction: ..dir..", counter: "..counter..", button: "..button")
end
```

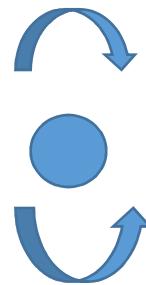
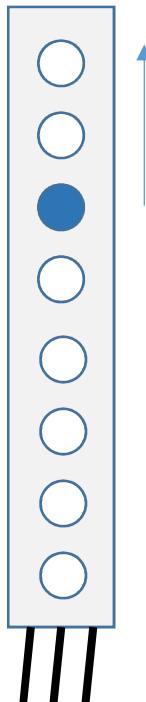
```
enc = encoder.attach(pio.GPIO19, pio.GPIO21, pio.GPIO5, lecture)
```

Ou GPIO4
(system0)

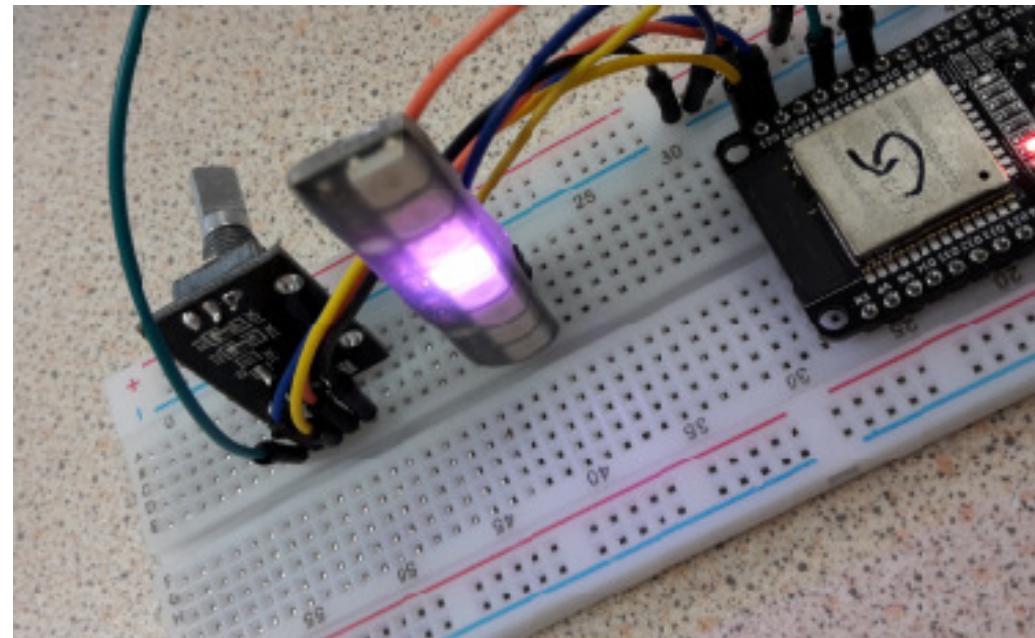
```
/ > function lecture(dir, counter, button)
>>     print("direction: ..dir..", counter: "..counter..", button: "..button")
>> end
/ > enc = encoder.attach(pio.GPIO19, pio.GPIO21, pio.GPIO5, lecture)
/ > direction: -1, counter: -1, button: 0
direction: -1, counter: -2, button: 0
direction: -1, counter: -3, button: 0
direction: 1, counter: -2, button: 0
direction: 1, counter: -1, button: 0
direction: 1, counter: 0, button: 0
direction: 1, counter: 1, button: 0
direction: 1, counter: 2, button: 0
direction: 1, counter: 3, button: 0
```

ESP32 + Lua RTOS

Réalisez le montage + code suivant : **deplace.lua**



Tourner le bouton dans un sens fait monter la led allumée
Et tourner dans l'autre sens la fait descendre



- Rappels

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8)
```

```
neo:setPixel(numero, r,v,b)
```

```
neo:update()
```

```
function action(dir, counter, button)
```

```
...
```

```
end
```

```
enc = encoder.attach(pio.GPIO19, pio.GPIO21, pio.GPIO5,  
action)
```

Ou GPIO4
(system0)

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18, 8)

position = 0
position_precedente = 0
neo:setPixel(position, 20,0,20)
neo:update()

function action(direction, compteur, bouton)
    if direction == -1 and position > 0 then position = position -1
        elseif direction == 1 and position < 7 then position = position +1
    end
    neo:setPixel(position_precedente, 0,0,0)
    neo:setPixel(position, 20, 0, 20)
    position_precedente = position
    neo:update()
end

enc = encoder.attach(pio.GPIO19, pio.GPIO21, pio.GPIO5, action)
```

deplace.lua

Ou GPIO4
(system0)

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18, 8)
```

deplace_rgb.lua

```
position = 0  
position_precedente = 0  
neo:setPixel(position, 20,0,0)  
neo:update()
```

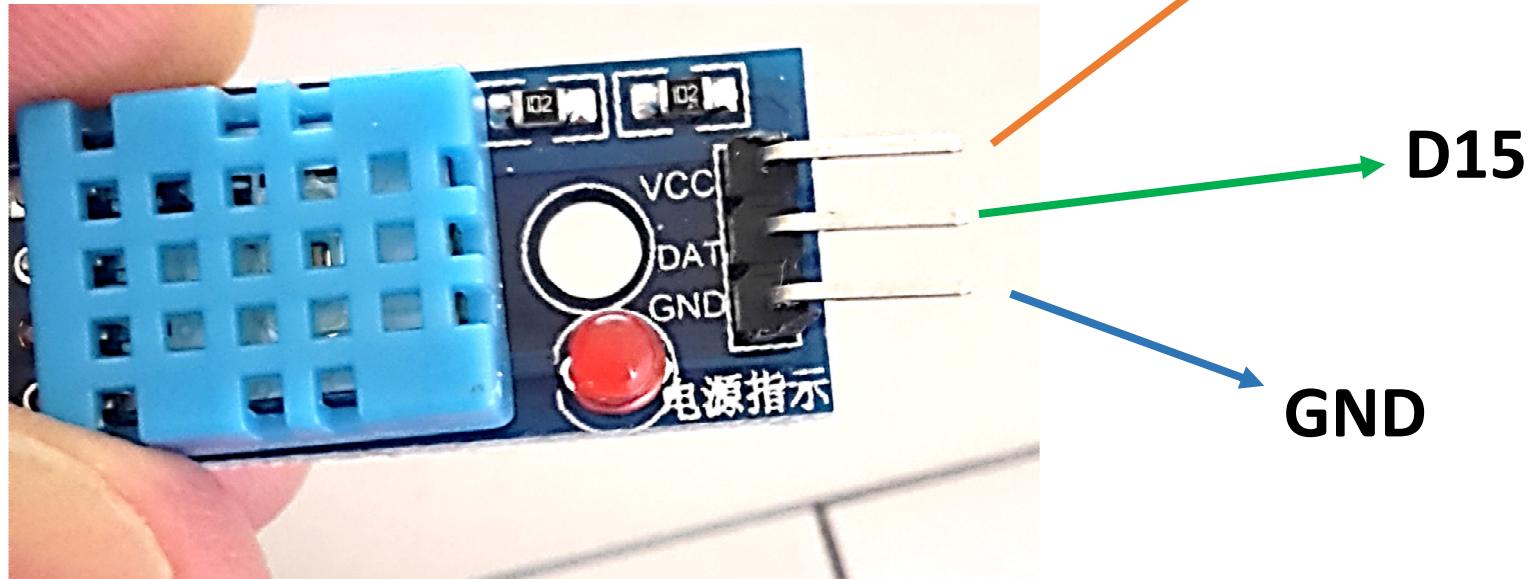
```
function action(direction, compteur, bouton)  
    if direction == -1 and position > 0 then position = position -1  
    elseif direction == 1 and position < 7 then position = position +1  
    end  
    neo:setPixel(position_precedente, 0,0,0)  
r,v,b = wheelRGB( (position*255)//8 )  
neo:setPixel(position, r//20, v//20, b//20)  
    position_precedente = position  
    neo:update()  
end
```

Ou GPIO4
(system0)

```
enc = encoder.attach(pio.GPIO19, pio.GPIO21, pio.GPIO5, action)
```

ESP32 + Lua RTOS

Capteur de température et humidité : DHT11



```
s = sensor.attach("DHT11", pio.GPIO15)
while true do
    print("temp: "..s:read("temperature").." humidite : "..s:read("humidity"))
    tmr.delayms(500)
end
```

Notez l'humidité (par exemple 50%)

ESP32 + Lua RTOS

Capteur de température et humidité : DHT11

```
s = sensor.attach("DHT11", pio.GPIO15)
while true do
    print("temp: "..s:read("temperature").." humidite : "..s:read("humidity"))
    tmr.delayms(500)
end
```

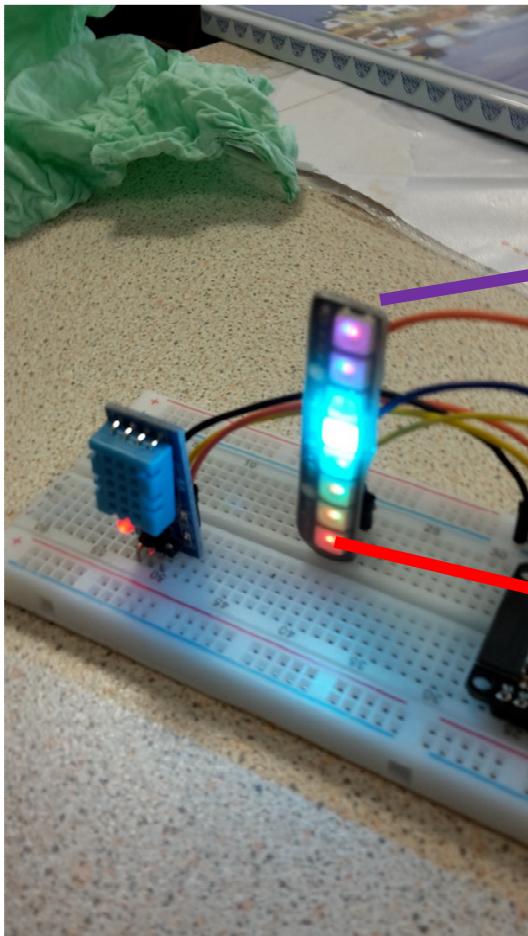
```
/ > dofile("temp.lua")
temp: 26.0 humidite : 63.0
temp: 26.0 humidite : 63.0
temp: 25.0 humidite : 55.0
temp: 25.0 humidite : 55.0
temp: 25.0 humidite : 53.0
temp: 25.0 humidite : 53.0
temp: 25.0 humidite : 50.0
temp: 25.0 humidite : 50.0
temp: 25.0 humidite : 49.0
temp: 25.0 humidite : 49.0
```

On peut augmenter l'humidité en posant le doigt sur le capteur (délicatement !...)

Notez la valeur stabilisée (par exemple 49%)

ESP32 + Lua RTOS

Affichage de l'humidité sur la bande de leds : valeurs entre 50 et 80



80% (et plus)

Valeur courante humidité

50% (et moins)

temp_led.lua
version sans thread

temp_led_th.lua
version avec thread

- Rappels

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18, 8)
```

```
neo:setPixel(numero, r,v,b)
```

```
neo:update()
```

```
s = sensor.attach("DHT11", pio.GPIO15)
```

```
s:read("humidity")
```

temp_led.lua
version sans thread

Valeur entière : math.floor()

temp_led_th.lua
version avec thread

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18, 8)
s = sensor.attach("DHT11", pio.GPIO15)
```

```
position = 0
position_ancienne = 0

while true do
    h = s:read("humidity")
    position = math.floor(((h-50)/(80-50))*8)
    if position < 0 then position = 0
        elseif position > 7 then position = 7
    end

    neo:setPixel(position_ancienne, 0,0,0)
    neo:setPixel(position, 20,0,0)
    position_ancienne = position
    neo:update()
end
```

temp_led.lua
version sans thread



Boot LUARTOS

Lancement de **config.lua**, puis **system.lua**

Dans **system.lua** : appel de `dofile("console.lua")`
rajout de fonctions utilitaires perso

Désactivez en commentant :

```
-- dofile("console.lua")
print("")
```

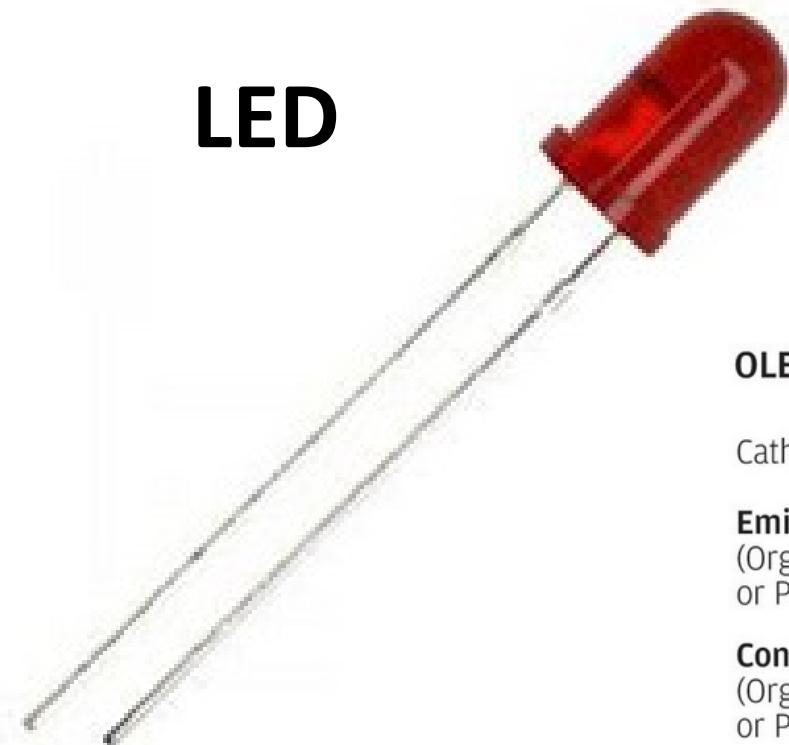
print("IOT - config console G.Ménier - Université de Bretagne Sud")

> reboot

Vérifiez que **ledon()** et **ledoff()** ne marchent plus.

ESP32 + Lua RTOS

LED



OLED

OLED structure

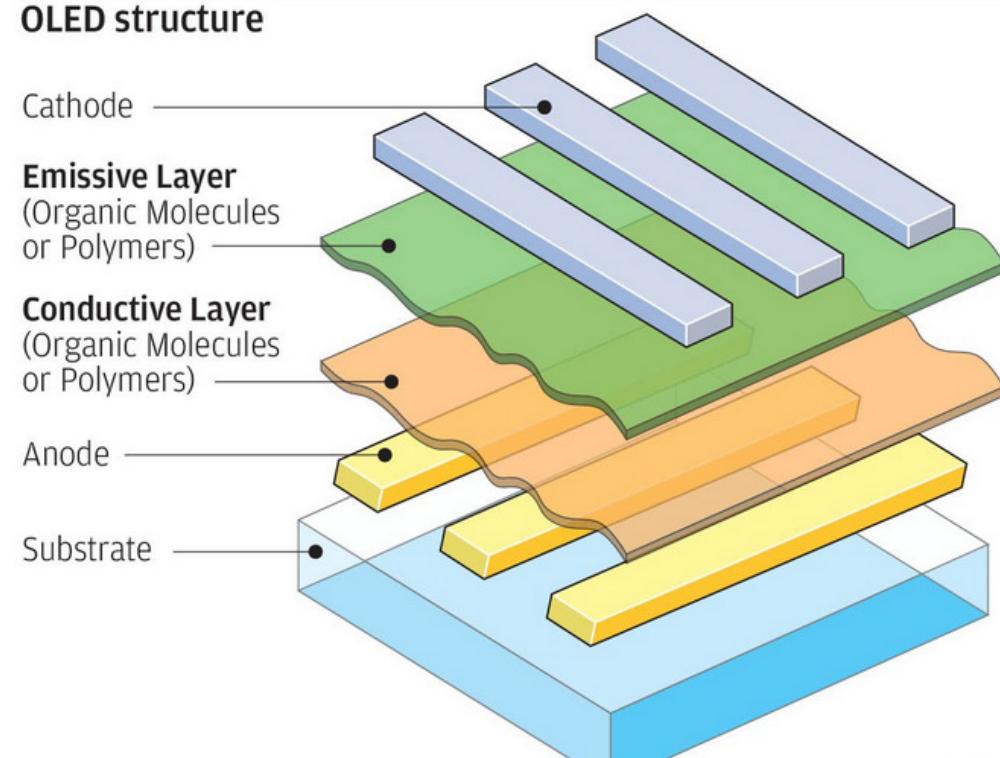
Cathode

Emissive Layer
(Organic Molecules
or Polymers)

Conductive Layer
(Organic Molecules
or Polymers)

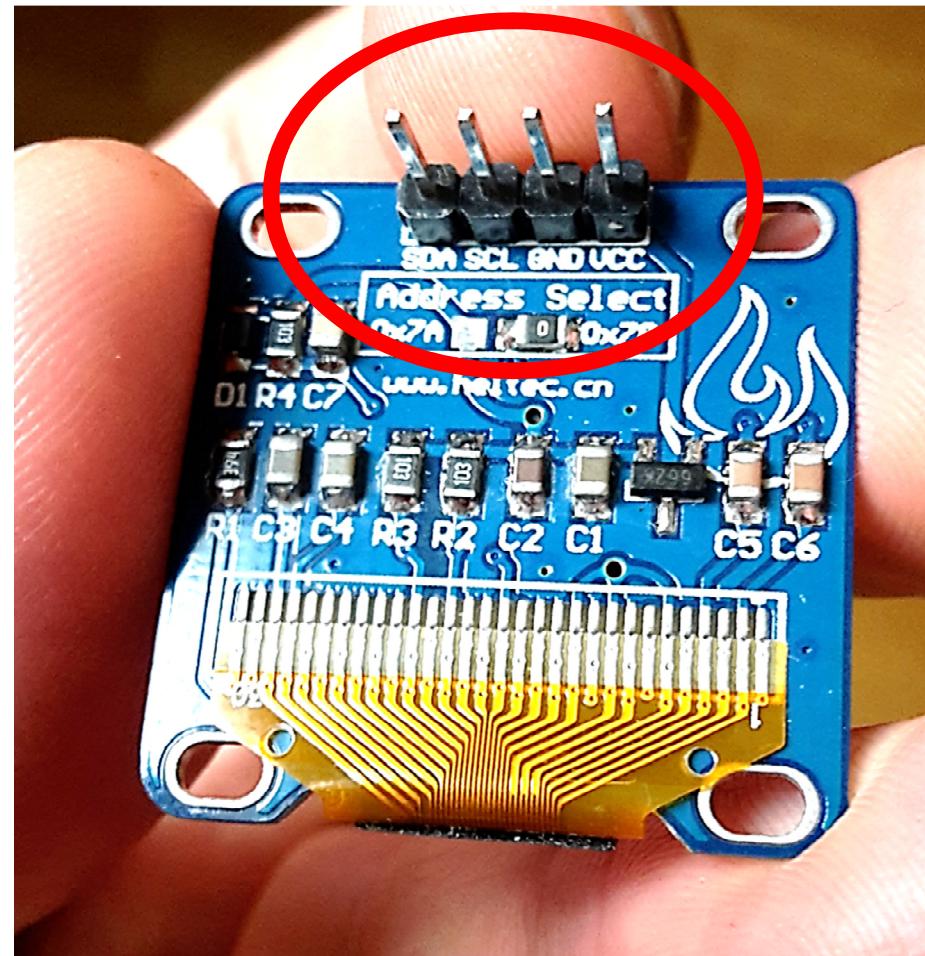
Anode

Substrate



ESP32 + Lua RTOS

Mini Écran OLED : 128x64 (8192)



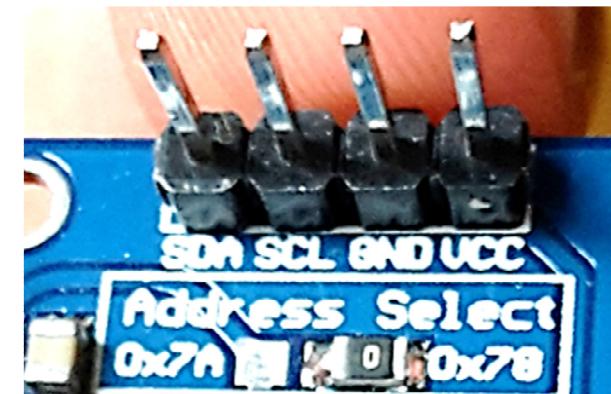
ESP32 + Lua RTOS

On ne dispose pas de 8192 pattes pour contrôler chaque micro led ...

Protocole de communication spécial IOT : i2c

Inter Integrated Circuit (Philips)

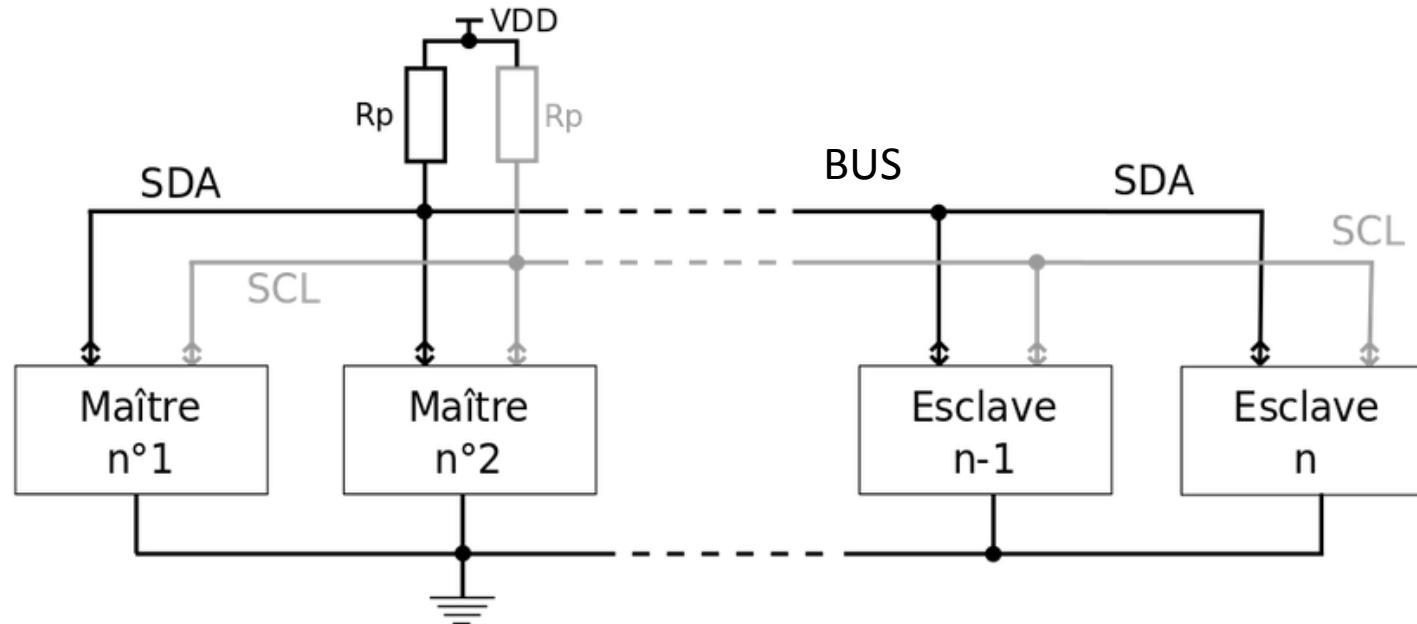
TWI : Two Wire Interface



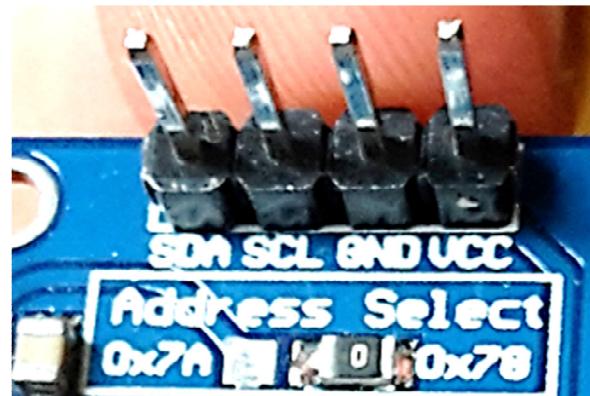
Deux fils :

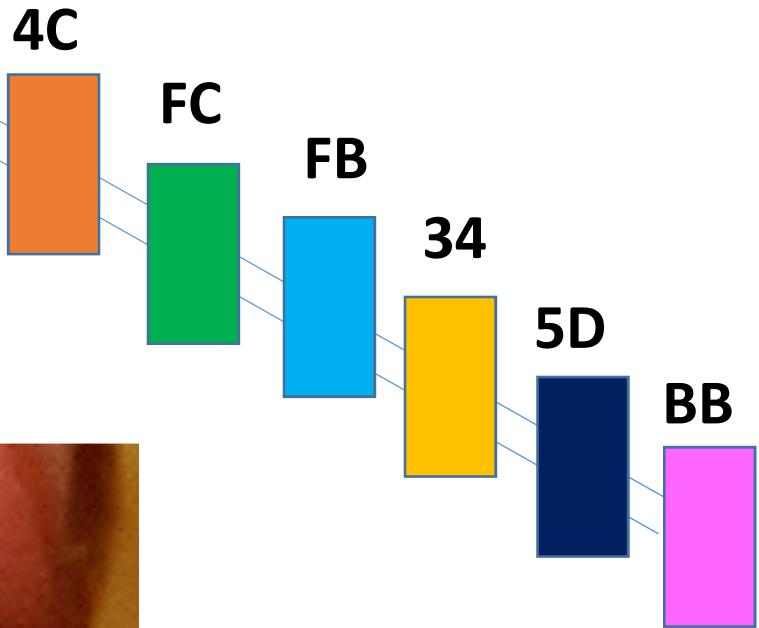
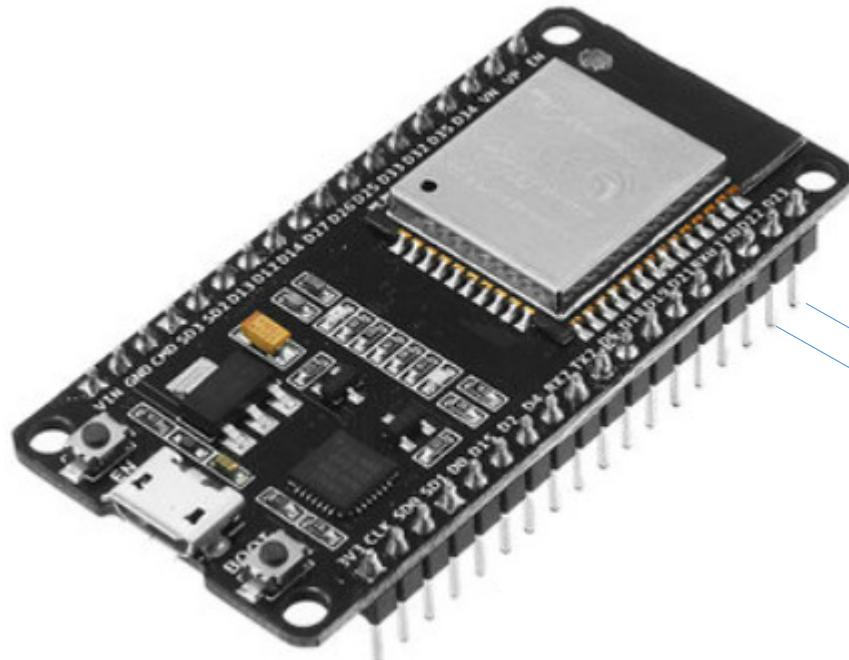
SDA : signal de données (deux sens)

SCL : horloge (cadence le transfert d'information)



Chaque interface possède un numéro (unique) qui sert à l'identifier
Sur 7 bits : 128 périphériques possibles en //



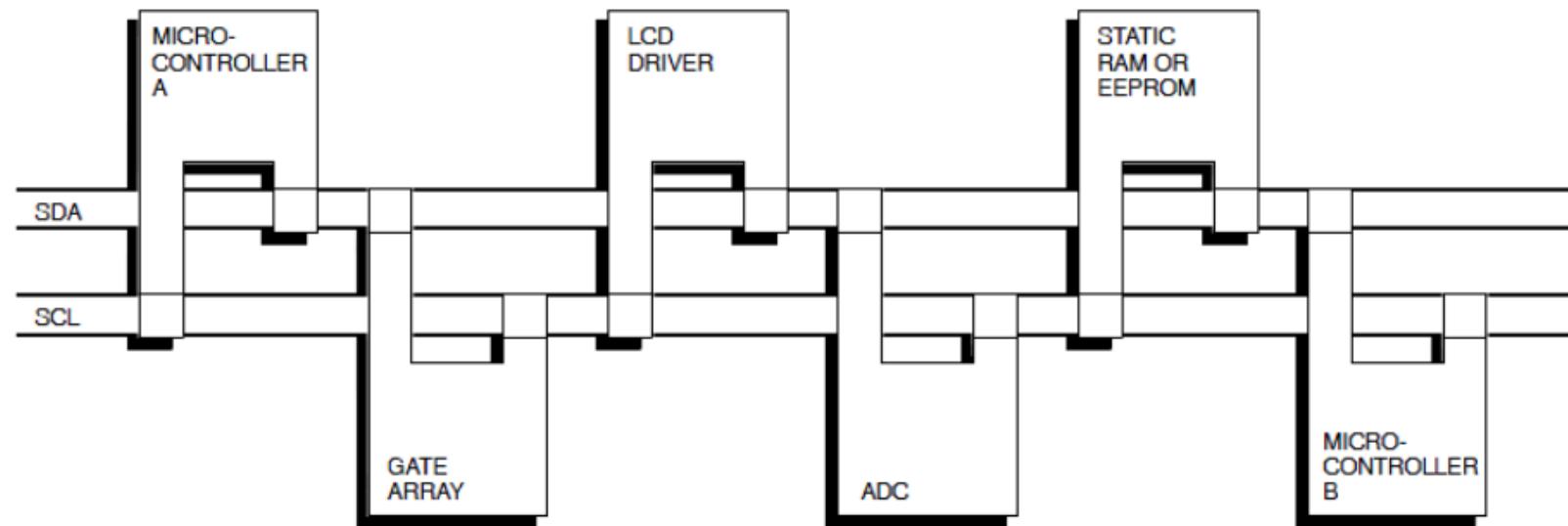


en rajoutant évidemment
GND et +

Adresse i2C
Résistance pour choisir
entre deux adresses

ESP32 + Lua RTOS

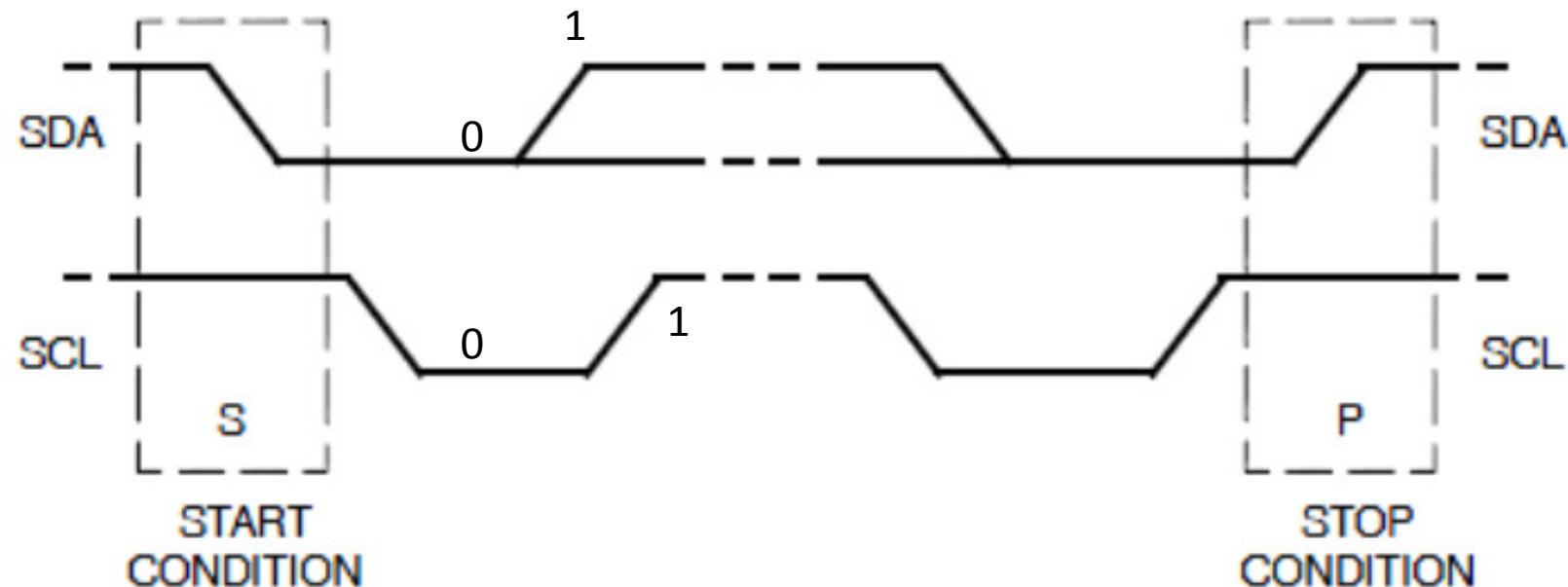
Interface de communication Philips



Genelaix

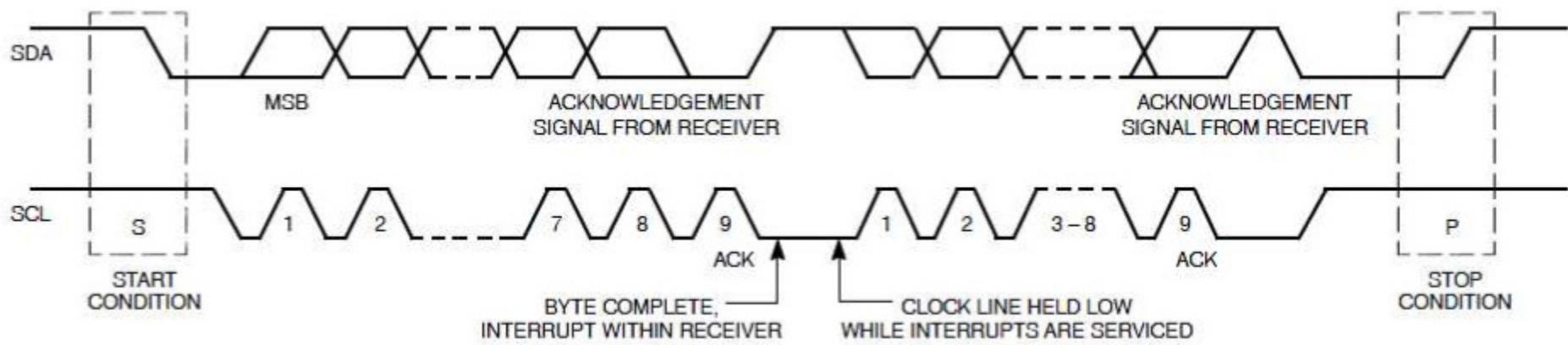
ESP32 + Lua RTOS

Interface de communication Philips



100khz – 400 khz

ESP32 + Lua RTOS



Transmission sur le front descendant de scl

On ne lit la donnée que quand scl passe de 1 à 0

Nota bene : pwm et neopixel

ESP32 + Lua RTOS

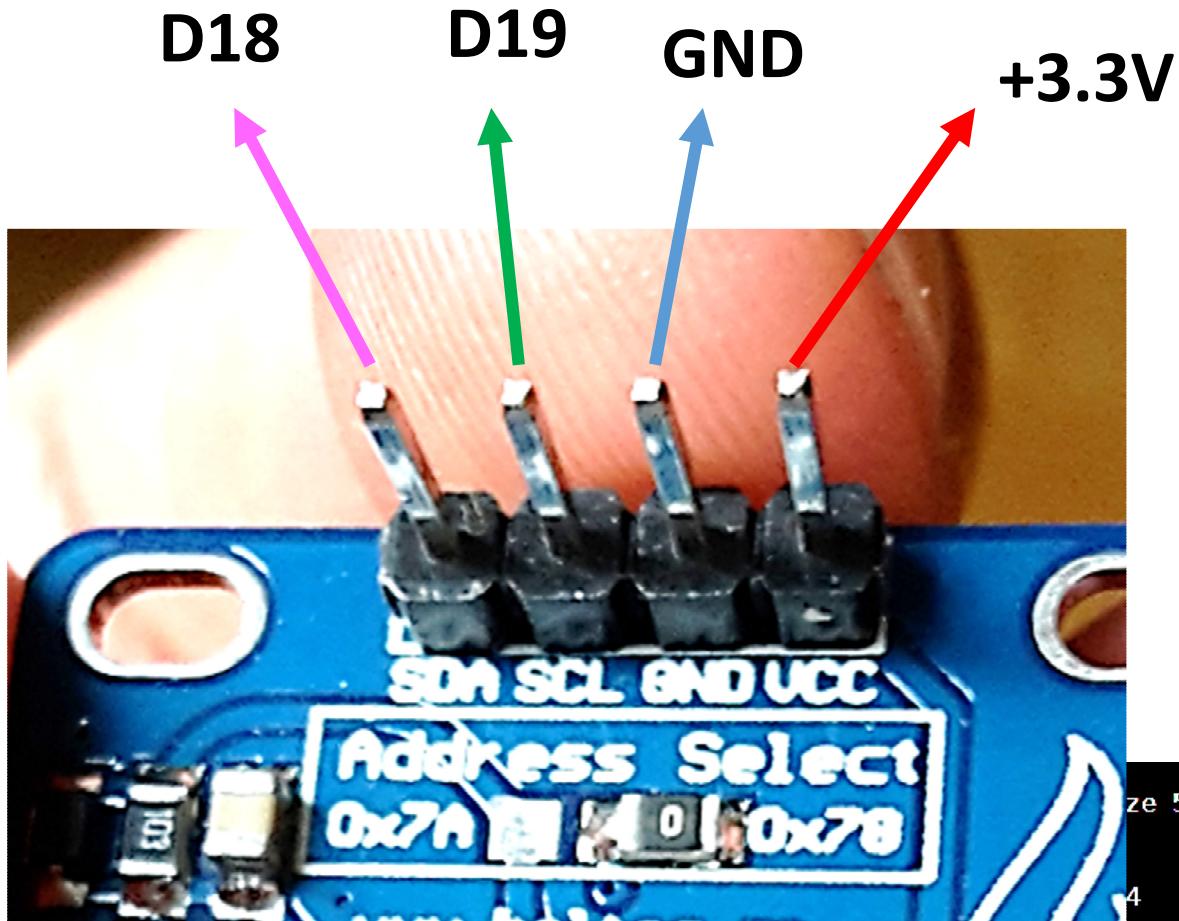
Acknowledge



- Le maître libère la ligne SDA
- L'esclave force la ligne SDA au niveau bas (trait gras)
- Le maître envoie une impulsion sur l'horloge SCL
- Lorsque l'impulsion retombe à zéro, l'esclave libère SDA

Diou

ESP32 + Lua RTOS



```
Executing /system.lua ...
i2c0 at pins scl=GPIO23/sdc=GPIO22
OLED SSD1306 at i2c0

IOT - Gildas Menier - Universite de Bretagne Sud
Executing /autorun.lua ...
```

ESP32 + Lua RTOS

D18 : SDA
D19 : SCL

ssd1306 : protocole de communication (qui utilise i2c : interface de communication)

10.1.9 Entire Display ON (A4h/A5h)

A4h command enable display outputs according to the GDDI. If A5h command is issued, then by using A4h command, the In other words, A4h command resumes the display from entity. A5h command forces the entire display to be “ON”, regardless.

10.1.10 Set Normal/Inverse Display (A6h/A7h)

This command sets the display to be either normal or inverse. “ON” pixel while in inverse display a RAM data of 0 indicate

10.1.11 Set Multiplex Ratio (A8h)

This command switches the default 63 multiplex mode to any output pads COM0~COM63 will be switched to the correspo

10.1.12 Set Display ON/OFF (AEh/AFh)

These single byte commands are used to turn the OLED pane. When the display is ON, the selected circuits by Set Master C. When the display is OFF, those circuits will be turned OFF state and high impedance state, respectively. These command

- o AEh : Display OFF

4 BLOCK DIAGRAM

5 DIE PAD FLOOR PLAN

6 PIN ARRANGEMENT.....

6.1 SSD1306TR1 PIN ASSIGNMENT

7 PIN DESCRIPTION

8 FUNCTIONAL BLOCK DESCRIPTIONS.....

8.1 MCU INTERFACE SELECTION.....

8.1.1 MCU Parallel 6800-series Interface.....

8.1.2 MCU Parallel 8080-series Interface.....

8.1.3 MCU Serial Interface (4-wire SPI).....

8.1.4 MCU Serial Interface (3-wire SPI).....

8.1.5 MCUI²C Interface.....

8.2 COMMAND DECODER

8.3 OSCILLATOR CIRCUIT AND DISPLAY TIME GENERATOR.....

8.4 FR SYNCHRONIZATION

8.5 RESET CIRCUIT

8.6 SEGMENT DRIVERS / COMMON DRIVERS

8.7 GRAPHIC DISPLAY DATA RAM (GDDRAM).....

8.8 SEG/COM DRIVING BLOCK

8.9 POWER ON AND OFF SEQUENCE

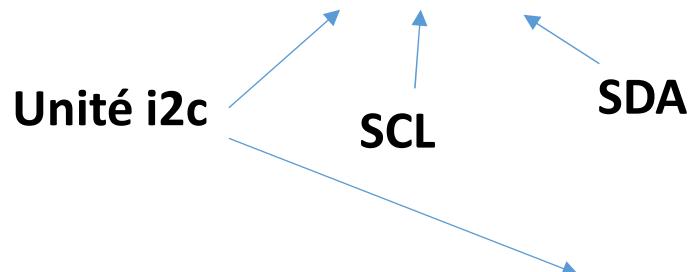
9 COMMAND TABLE

ESP32 + Lua RTOS

D18 : SDA
D19 : SCL

ssd1306 : protocole de communication
(qui utilise i2c : interface de communication)

i2c.setpins(0,19,18)



Vitesse clock scl
Ici 400Khz

```
com = i2c.attach(i2c.I2C0, i2c.MASTER, 400000)
com:start()
com:address(devAddress, false)
com:write(0x00, address, valeur)
-- ou bien com:write(0x00, commande)
com:stop()
```

ESP32 + Lua RTOS

D18 : SDA
D19 : SCL

Comment trouver l'adresse I₂C ?

Documentation, mais surtout et aussi :

Scan I₂C : on essaye toutes les adresses I₂C sur le bus

com:start()

com:address(i, false)

com:stop()

Normalement lève une exception si erreur

N'afficher QUE les cas sans erreur ? Pas de try/catch en lua

Mais...

ESP32 + Lua RTOS

D18 : SDA
D19 : SCL

Soit une fonction lua F :

If pcall(F) then -- pas de pb

...

else -- erreur

...

end

-- pcall pour 'protected' call

-- status, result = pcall(F)

-- vrai/faux, erreur = pcall(F)

ESP32 + Lua RTOS

D18 : SDA
D19 : SCL

Brancher l'écran et réaliser un programme Lua qui détermine son adresse I2C.

```
i2c.setpins(1,18,19)
```

```
print("Scan adresses I2C")
```

```
ic = i2c.attach(i2c.I2C1, i2c.MASTER)
```

```
function catch(what)
```

```
    return what[1]
```

```
end
```

```
function try(what)
```

```
    status, result = pcall(what[2])
```

```
    if not status then
```

```
        what[3](result)
```

```
    else what[1]()
```

```
    end
```

```
    return result
```

```
end
```

```
for i=0,127 do
```

```
    try {
```

```
        function()
```

```
            print(string.format("trouvé # %x", i))
```

```
        end,
```

```
        function()
```

```
            ic:start()
```

```
            ic:address(i, false)
```

```
            ic:stop()
```

```
        end,
```

```
    catch {
```

```
        function(error)
```

```
    end
```

```
}
```

```
end
```

LUA

Scala

Javascript

Scan d'adresse i2c

```
/ > scani2c.lua
Scan adresses I2C
i2c1 at pins scl=GPIO19/scl=GPIO18
trouv # 3c
/ >
```

ESP32 + Lua RTOS

D18 : SDA
D19 : SCL

ssd1306 : adresse 0x3C

<https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf>

Display_off

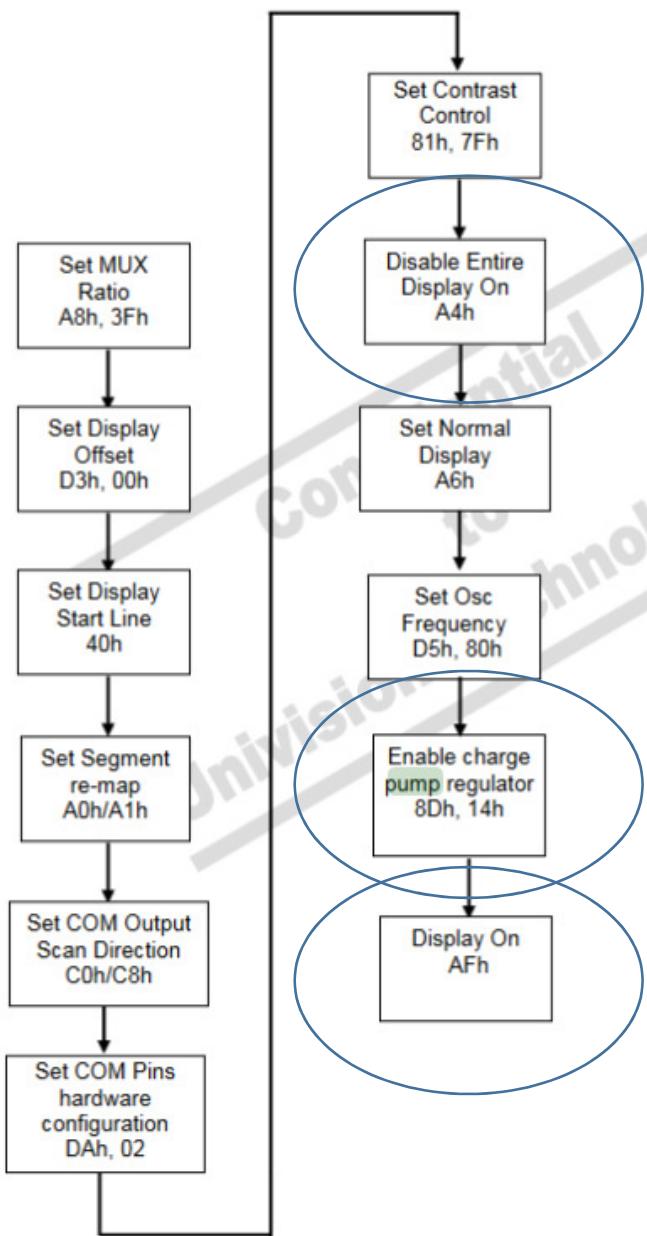
Charge Pump (0x14)

Display_on

**Creez une fonction pour
envoyer un message**

```
i2c.setpins(0,19,18)
com = i2c.attach(i2c.I2C0, i2c.MASTER, 400000)
com:start()
com:address(0x3C, false)
com:write(0x00, address, valeur)
-- ou bien com:write(0x00, commande)
com:stop()
```

Figure 2 : Software Initialization Flow Chart



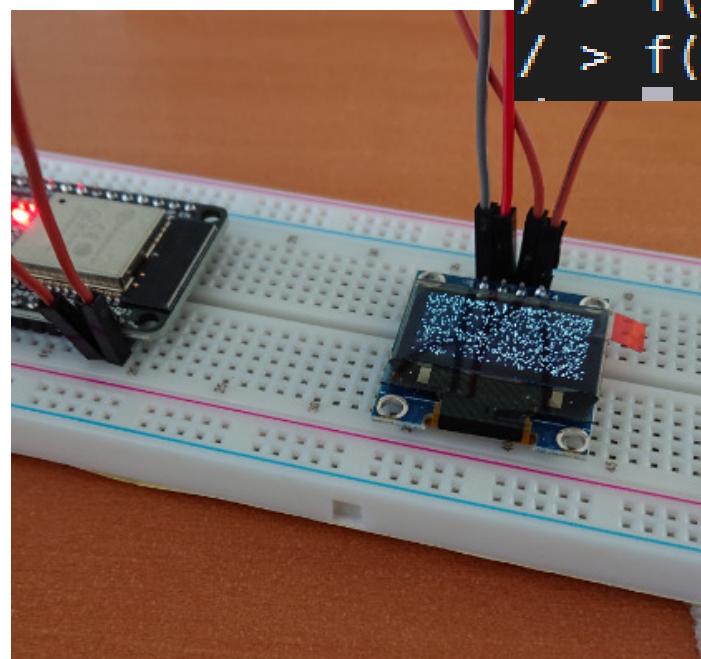
```

i2c.setpins(1,22,23)
com = i2c.attach(i2c.I2C1,i2c.MASTER)

f = function(cmd)
    com:start()
    com:address(0x3C, false)
    com:write(0x00, cmd)
    com:stop()
end

    / > f(0xA4)
    / > f(0x8D)
    / > f(0x14)
    / > f(0xAF)

```



ESP32 + Lua RTOS

D18 : SDA
D19 : SCL

Alimenter écran

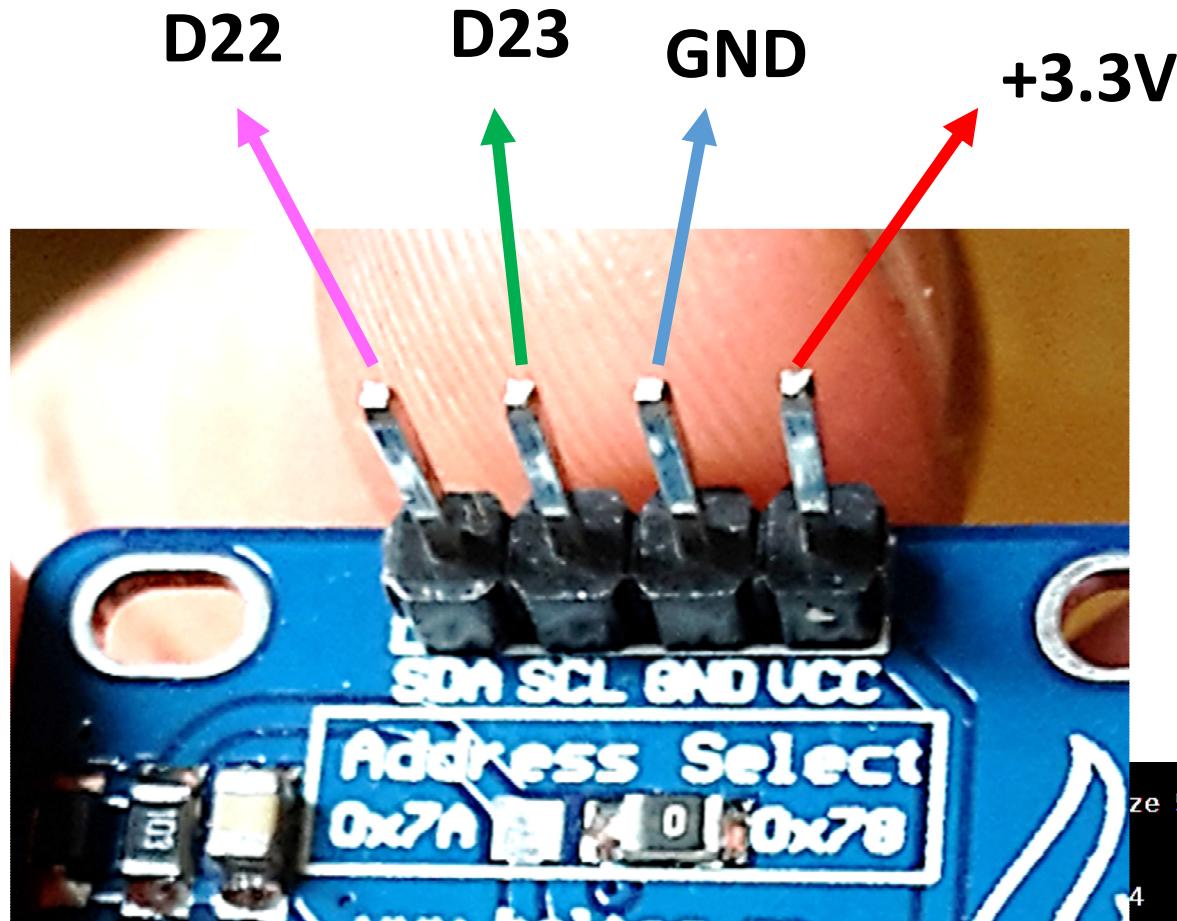
Mais aussi : configurer pour l'affichage
indiquer l'adresse mémoire de début
indiquer la taille
indiquer le mode d'adressage
la gestion de la tension
vider la mémoire
...

Afficher un point :
préparer le mode d'écriture
calculer l'adresse mémoire du point
offset octet
récupérer l'octet présent
ou logique
placer le nouvel octet en mémoire
etc

Afficher un caractère :
calculer l'adresse
récupérer en mémoire hôte
codage binaire
placer en mémoire or
etc..

ESP32 + Lua RTOS

Réactiver console.lua dans system.lua



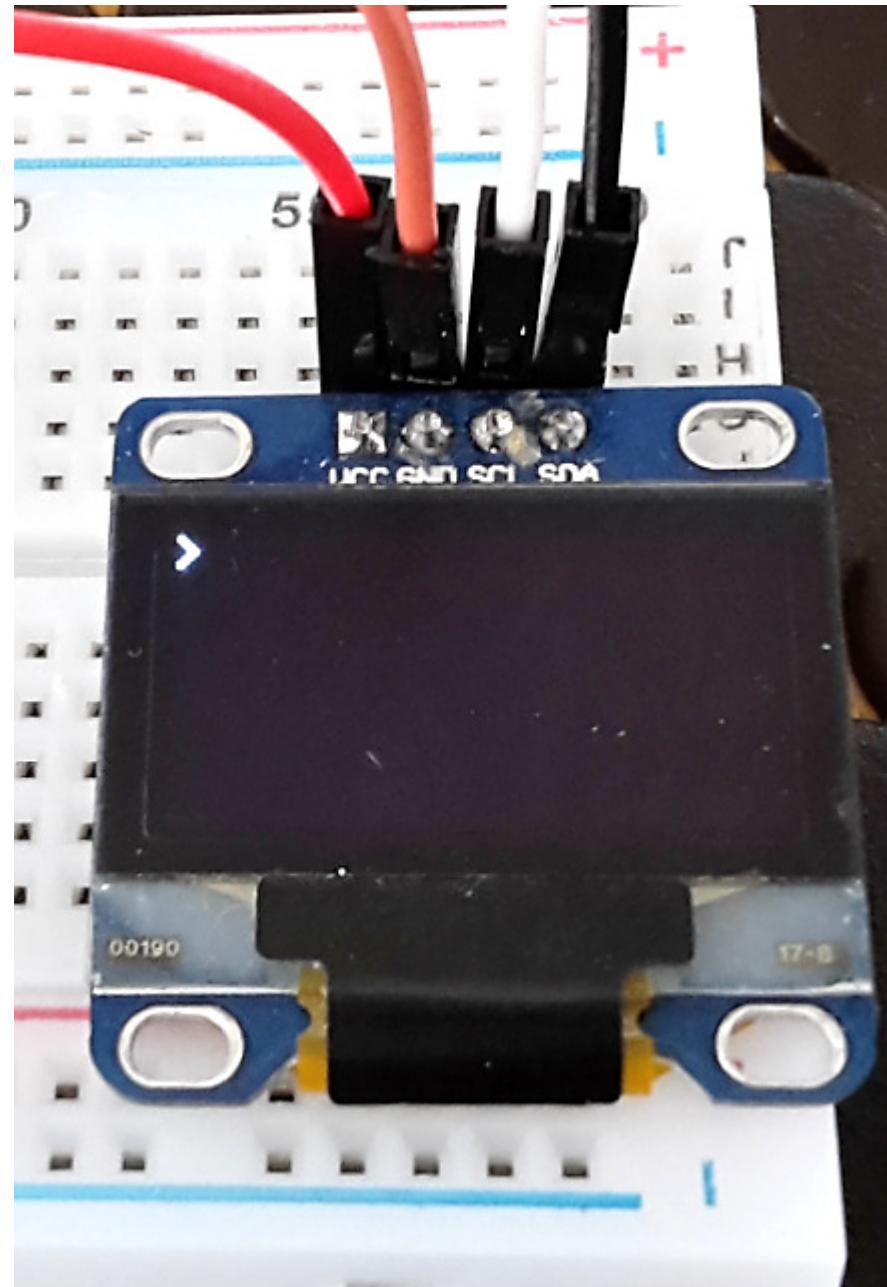
```
Executing /system.lua ...
i2c0 at pins scl=GPIO23/sdc=GPIO22
OLED SSD1306 at i2c0
```

```
IOT - Gildas Menier - Universite de Bretagne Sud
Executing /autorun.lua ...
```

ESP32 + Lua RTOS

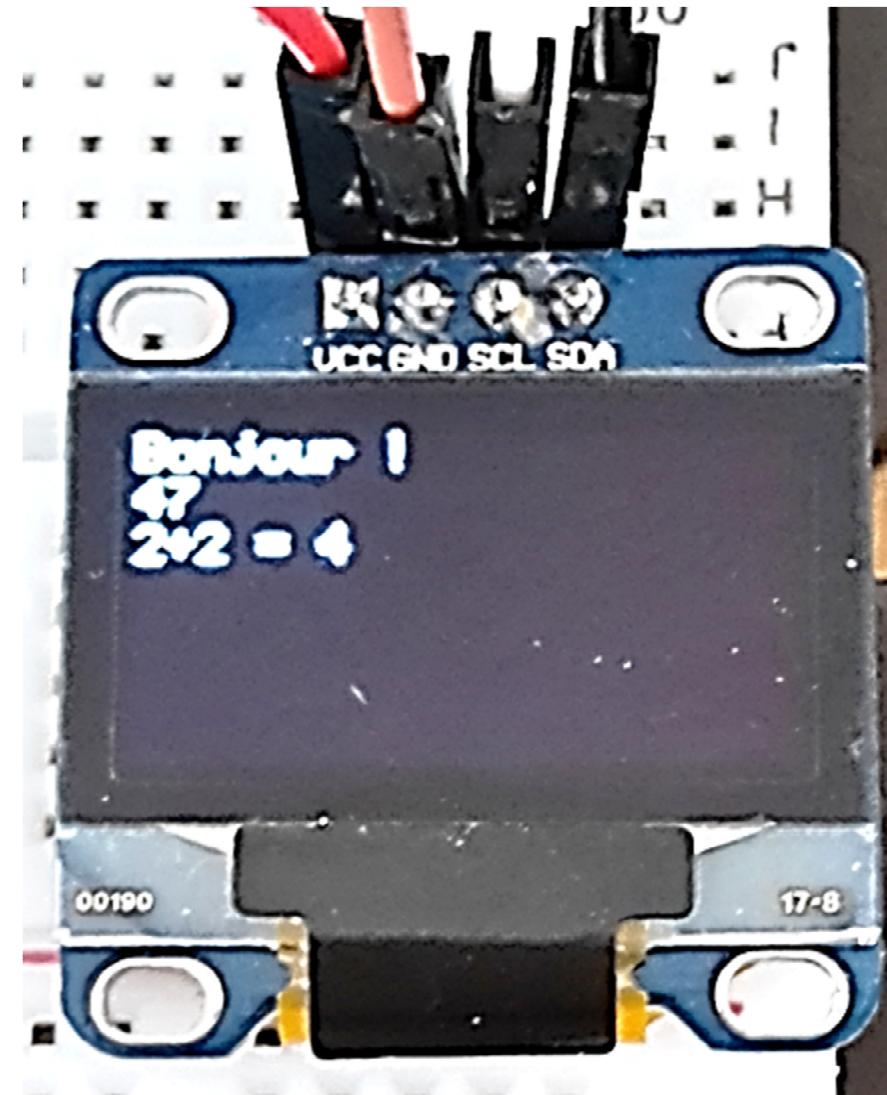
L'esp32 est équipé de primitives de contrôle pour gérer la communication i2C

Ce sont des fonctions LUA



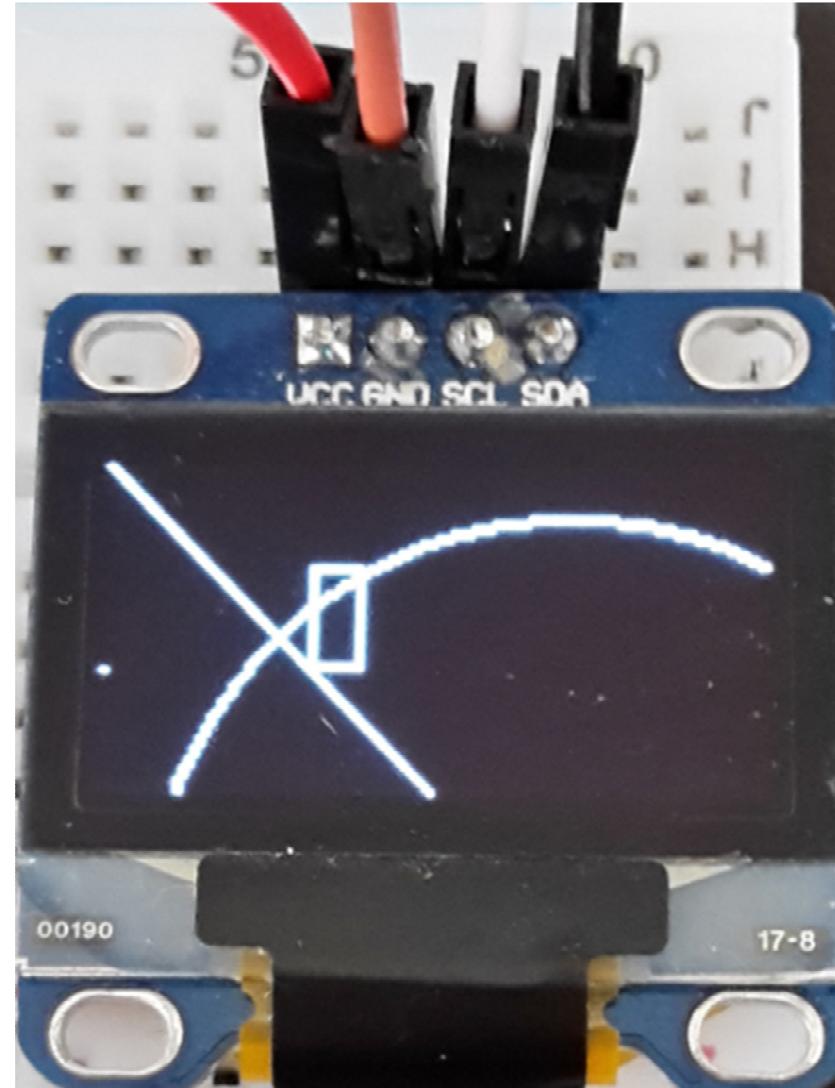
ESP32 + Lua RTOS

```
cls()  
console("Bonjour !")  
console(47)  
console("2+2.."..2+2)
```



ESP32 + Lua RTOS

```
cls()  
gdisplay.line({0,0},{150,150})  
gdisplay.putpixel(0,40)  
gdisplay.off()  
gdisplay.on()  
gdisplay.invert(true)  
gdisplay.invert(false)  
gdisplay.rect({40, 20}, 10, 20)  
gdisplay.circle({90,90}, 80)
```



ESP32 + Lua RTOS

Affichez la température toutes les 20s



latest

Search docs

Get Started

Get Started (CMake Preview)

API Reference

Wi-Fi

Mesh

Bluetooth

Ethernet

Peripherals

Protocols

Provisioning

Storage

System

FreeRTOS

FreeRTOS Additions

Heap Memory Allocation

[Read the Docs](#)

v: latest ▾

Sleep Modes

Overview

ESP32 is capable of light sleep and deep sleep power saving modes.

In light sleep mode, digital peripherals, most of the RAM, and CPUs are clock-gated, and supply voltage is reduced. Upon exit from light sleep, peripherals and CPUs resume operation, their internal state is preserved.

In deep sleep mode, CPUs, most of the RAM, and all the digital peripherals which are clocked from APB_CLK are powered off. The only parts of the chip which can still be powered on are: RTC controller, RTC peripherals (including ULP coprocessor), and RTC memories (slow and fast).

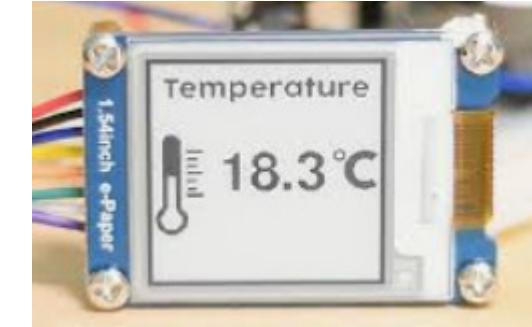
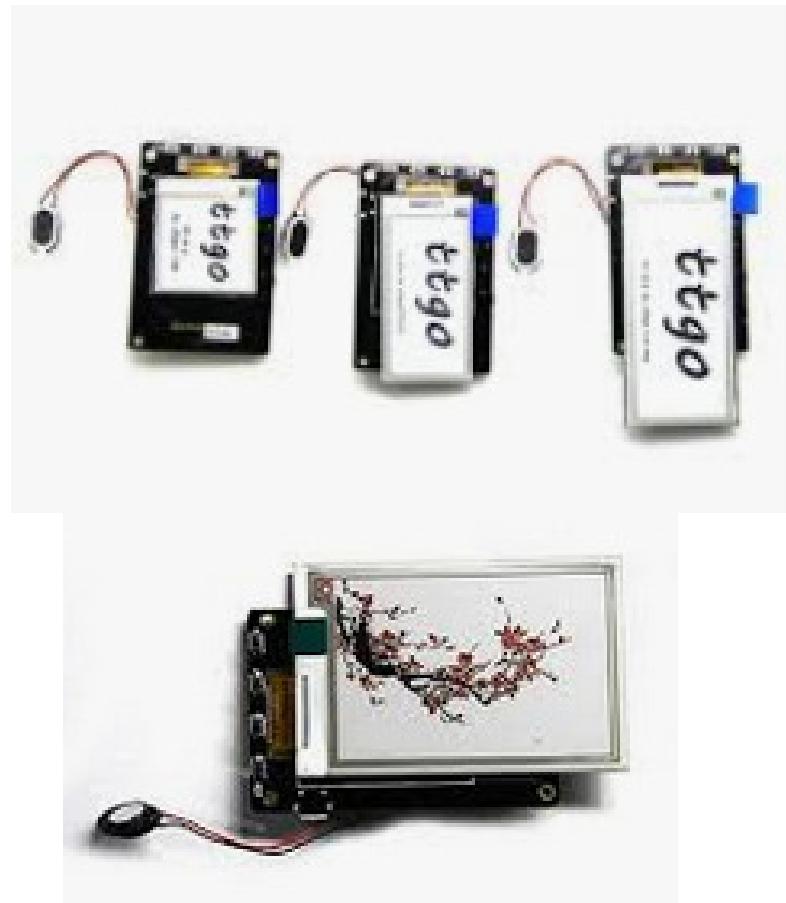
Wakeup from deep and light sleep modes can be done using several sources. These sources can be combined, in this case the chip will wake up when any one of the sources is triggered. Wakeup sources can be enabled using `esp_sleep_enable_X_wakeup` APIs and can be disabled using `esp_sleep_disable_wakeup_source()` API. Next section describes these APIs in detail. Wakeup sources can be configured at any moment before entering light or deep sleep mode.

Additionally, the application can force specific powerdown modes for the RTC peripherals and RTC memories using `esp_sleep_pd_config()` API.

os.sleep(nbSeconds)

ESP32 + Lua RTOS

os.sleep(nbSecondes)

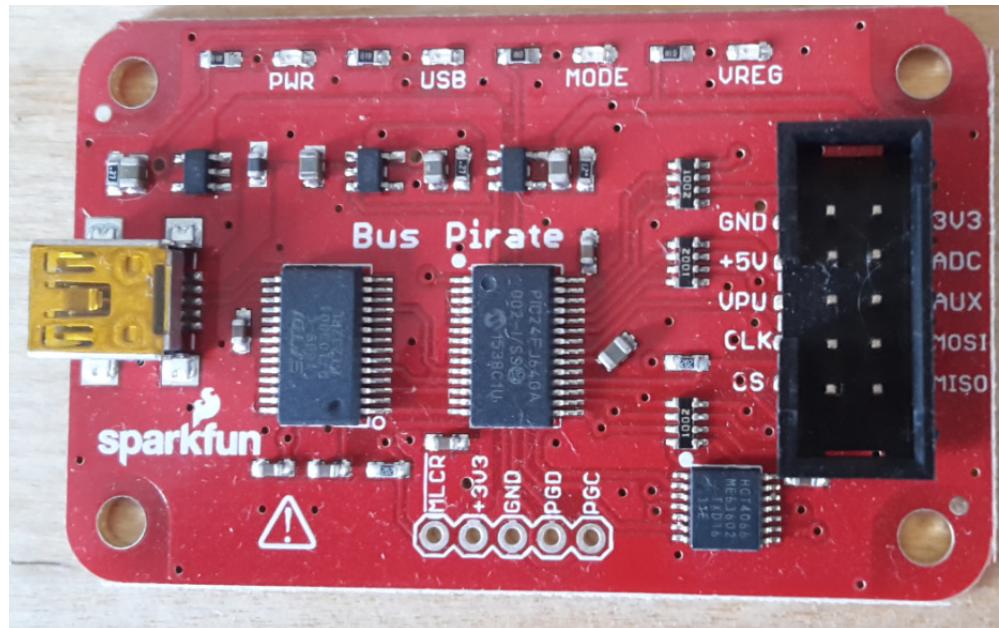


ESP32 + Lua RTOS

- Bilan pour l'instant
- Internet des objets
 - Quels objets ?
 - Microcontrôleurs
 - Capteurs
 - Indicateurs / effecteurs
 - Langage 'léger' : LUA / Javascript
 - Système d'exploitation temps réel
 - Liens entre processeur et le monde extérieur
 - GPIO
 - Thread
 - Programmation asynchrone

ESP32 + Lua RTOS

- On peut équiper un PC de ports GPIO ou d'interfaces i2c



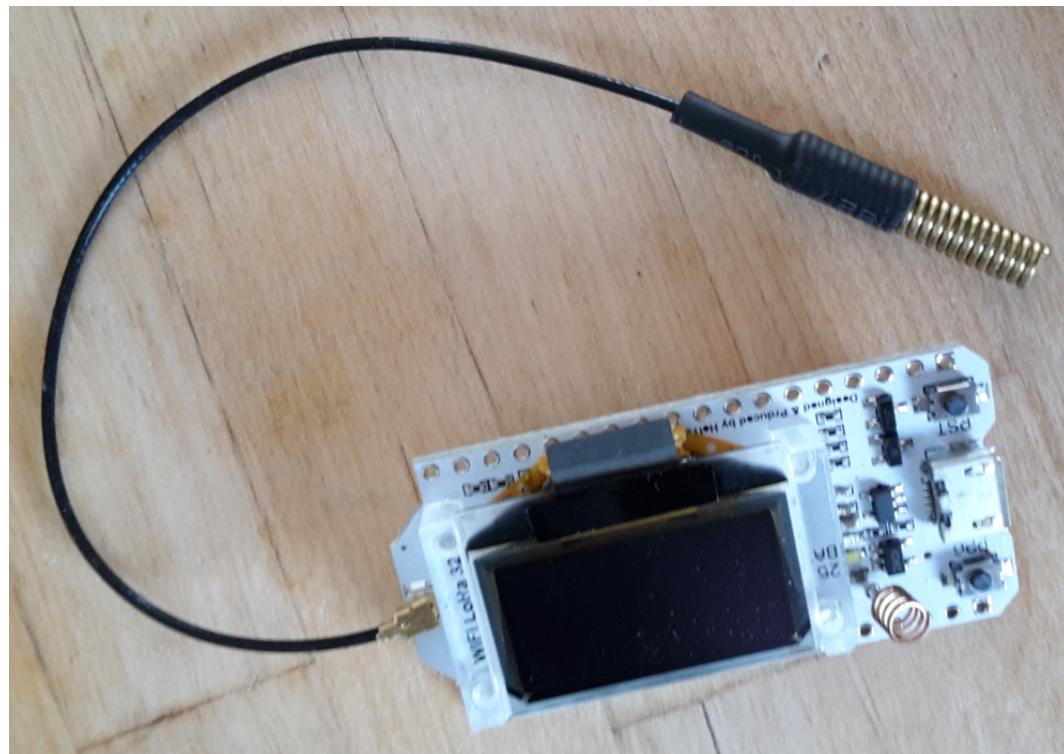
- Mais ça revient à les équiper d'un microcontrôleur...

ESP32 + Lua RTOS

Internet des objets / Internet Of Things

Connexion à Internet

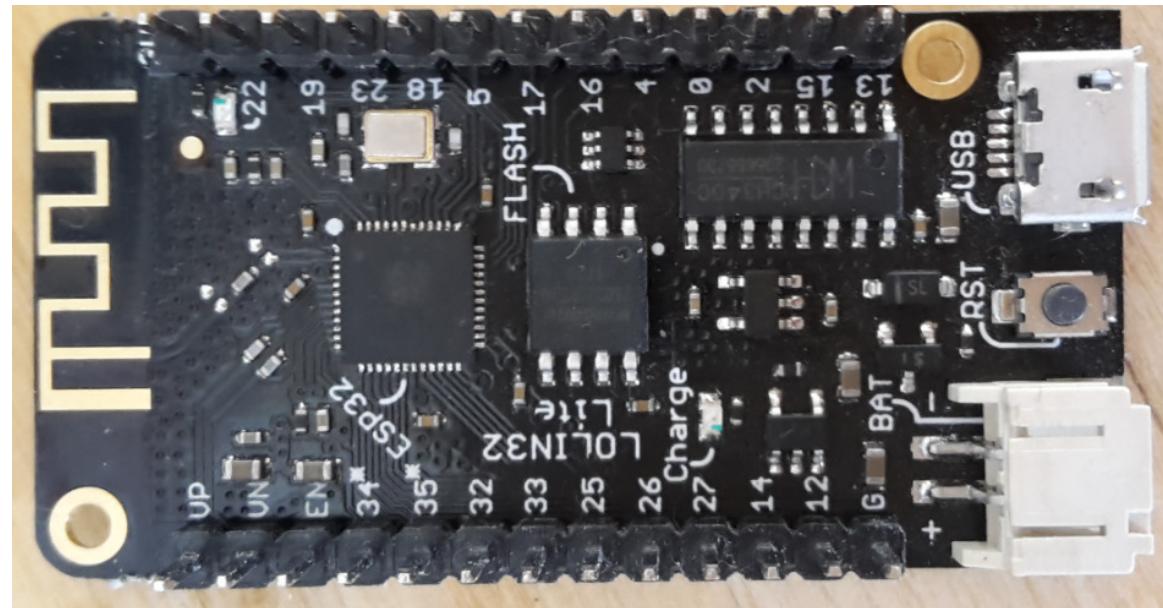
Possible par I2C (interface Ethernet)
mais également bluetooth et Wifi / LORA



ESP32 + Lua RTOS

Esp32 Wemos Lolin32

Antenne WIFI



ESP32 + Lua RTOS

1.2 Wi-Fi Key Features

- 802.11 b/g/n/e/i
- 802.11 n (2.4 GHz), up to 150 Mbps
- 802.11 e: QoS for wireless multimedia technology
- WMM-PS, UAPSD
- A-MPDU and A-MSDU aggregation
- Block ACK
- Fragmentation and defragmentation
- Automatic Beacon monitoring/scanning
 - 802.11 i security features: pre-authentication and TSN
 - Wi-Fi Protected Access (WPA)/WPA2/WPA2-Enterprise/Wi-Fi Protected Setup (WPS)
 - Infrastructure BSS Station mode/SoftAP mode
 - Wi-Fi Direct (P2P), P2P Discovery, P2P Group Owner mode and P2P Power Management
 - UMA compliant and certified
 - Antenna diversity and selection

1.3 BT Key Features

- Compliant with Bluetooth v4.2 BR/EDR and BLE specification
- Class-1, class-2 and class-3 transmitter without external power amplifier
- Enhanced power control
- +12 dBm transmitting power
- NZIF receiver with -97 dBm sensitivity
- Adaptive Frequency Hopping (AFH)
- Standard HCI based on SDIO/SPI/UART
- High-speed UART HCI, up to 4 Mbps
- BT 4.2 controller and host stack
- Service Discover Protocol (SDP)
- General Access Profile (GAP)
- Security Manage Protocol (SMP)
- ATT/GATT
- HID
- All GATT-based profile supported
- SPP-like GATT-based profile
- BLE Beacon
- A2DP/AVRCP/SPP, HSP/HFP, RFCOMM
- CVSD and SBC for audio codec
- Bluetooth Piconet and Scatternet

1.4.4 Security

- IEEE 802.11 standard security features are all supported, including WFA, WPA/WPA2 and WAPI
- Secure boot
- Flash encryption
- 1024-bit OTP, up to 768-bit for customers
- Cryptographic hardware acceleration:
 - AES
 - HASH (SHA-2) library
 - RSA
 - ECC
 - Random Number Generator (RNG)

ESP32 + Lua RTOS

Scanner Wifi : console

net.stat()

```
/ > net.stat()
wf: mac address 00:00:00:00:00:00
    ip address 0.0.0.0 / netmask 0.0.0.0
    gw address 0.0.0.0

en: mac address 00:00:00:00:00:00
    ip address 0.0.0.0 netmask 0.0.0.0
    gw address 0.0.0.0
```

affiche les informations de connexion
net.connected()

ESP32 + Lua RTOS

Scanner Wifi : console

net.wf.scan()

| | SSID | RSSI | AUTH | CH1 | CH2 |
|-------|-----------------|------|--------------|-----|-----|
| <hr/> | | | | | |
| | Freebox-48457C | -64 | WPA_PSK | 6 | 0 |
| | Livebox-2E1D | -68 | WPA_WPA2_PSK | 6 | 0 |
| | chateaufree | -71 | WPA_PSK | 1 | 0 |
| | FreeWifi | -71 | OPEN | 1 | 0 |
| | FreeWifi_secure | -71 | OPEN | 1 | 0 |
| | | -71 | WPA2_PSK | 1 | 0 |
| | Livebox-C4DE | -72 | WPA_WPA2_PSK | 11 | 0 |
| | orange | -72 | OPEN | 11 | 0 |
| | Livebox-ce80 | -89 | WPA_WPA2_PSK | 11 | 0 |

indique les réseaux wifi à portée
attention : un scan désactive les connexions courantes

ESP32 + Lua RTOS

```
w = net.wf.scan(true)
```

N'imprime pas le résultat : le résultat est stocké dans w
C'est une table LUA (format proche de JSON)

```
#w
```

Est la taille de la table

| / > net.wf.scan() | | | | | | |
|-------------------|-----------------|------|--------------|-----|-----|--|
| | SSID | RSSI | AUTH | CH1 | CH2 | |
| ----- | | | | | | |
| | Freebox-48457C | -64 | WPA_PSK | 6 | 0 | |
| | Livebox-2E1D | -68 | WPA_WPA2_PSK | 6 | 0 | |
| | chateaufree | -71 | WPA_PSK | 1 | 0 | |
| | FreeWifi | -71 | OPEN | 1 | 0 | |
| | FreeWifi_secure | -71 | OPEN | 1 | 0 | |
| | | -71 | WPA2_PSK | 1 | 0 | |
| | Livebox-C4DE | -72 | WPA_WPA2_PSK | 11 | 0 | |
| | orange | -72 | OPEN | 11 | 0 | |
| | Livebox-ce80 | -89 | WPA_WPA2_PSK | 11 | 0 | |

ESP32 + Lua RTOS

w[0] est le premier élément de la table

C'est également une table

```
/ > w[0]
table: 0x3ffebccc
/ >
```

```
/ > w[0].ssid
Freebox-48457C
/ > w[0].rssi
-64
/ > w[0].auth
2
/ >
```

| | SSID | RSSI | AUTH |
|-------|----------------|------|--------------|
| ----- | | | |
| | Freebox-48457C | -64 | WPA_PSK |
| | Livebox-2E1D | -68 | WPA_WPA2_PSK |
| | chateaufree | -71 | WPA_PSK |

ESP32 + Lua RTOS

Exercice :

réaliser un script lua qui affiche sur l'écran
OLED les bornes wifi accessibles (SSID + RSSI)
(les 4 premières) **scan_oled.lua**



Rappel :

```
w = net.wf.scan(true)
#w : taille
w[0].ssid
w[0].rss
cls()
console("2+2.."2+2)
```

scan_oled.lua

```
w = net.wf.scan(true)
nb = #w; if nb > 3 then nb = 3 end
cls()
for j=0,nb do
    console(w[j].ssid.." "..w[j].rssi)
end
```



```
cls()
w = net.wf.scan(true)
nb = #w; if nb > 3 then nb = 3 end
for j=0,nb do
    console(w[j].ssid.." "..w[j].rssi)
end
```

scan_oled_th.lua

Transformez votre code pour qu'il scanne les réseaux WIFI toutes les secondes avec un thread

Rappel :

```
function scan_wifi()
    while true do
        votre code ici
        tmr.delay(1)
    end
end
```

```
scan_thread = thread.start(scan_wifi)
```

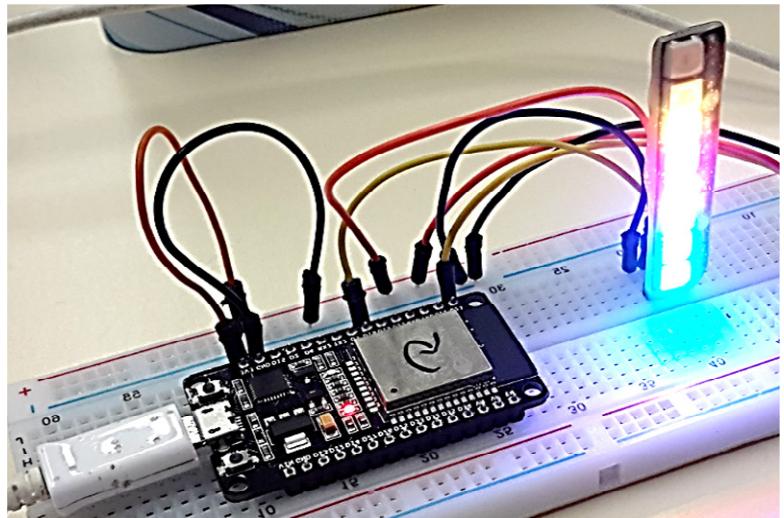
scan_oled_th.lua

```
function scan_wifi()
    while true do
        w = net.wf.scan(true)
        nb = #w; if nb > 3 then nb = 3 end
        cls()
        for j=0,nb do
            console(w[j].ssid.." "..w[j].rssi)
        end
        tmr.delay(1)
    end
end

scan_thread = thread.start(scan_wifi)
```

scan_oledrgb_th.lua

Maintenant, rajoutez un indicateur visuel de la puissance du signal



-30 et plus **Très bon signal**
(-20)

-80 et moins **très mauvais signal**
(-90 ...)

Rappel :

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8)  
neo:setPixel(numero, r,v,b)  
neo:update()
```

scan_oledrgb_th.lua

Rappel :

```
neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8)  
neo:setPixel(numero, r,v,b)  
neo:update()
```

Comment convertir le signal rssi en position sur la barette led (0-7) ?

Hypothèse : rssi est entre -80 et -30

rssи + 80 est entre -80+80 et -30+80

rssи + 80 est entre 0 et 50

(rssи +80)/50 est entre 0 et 1

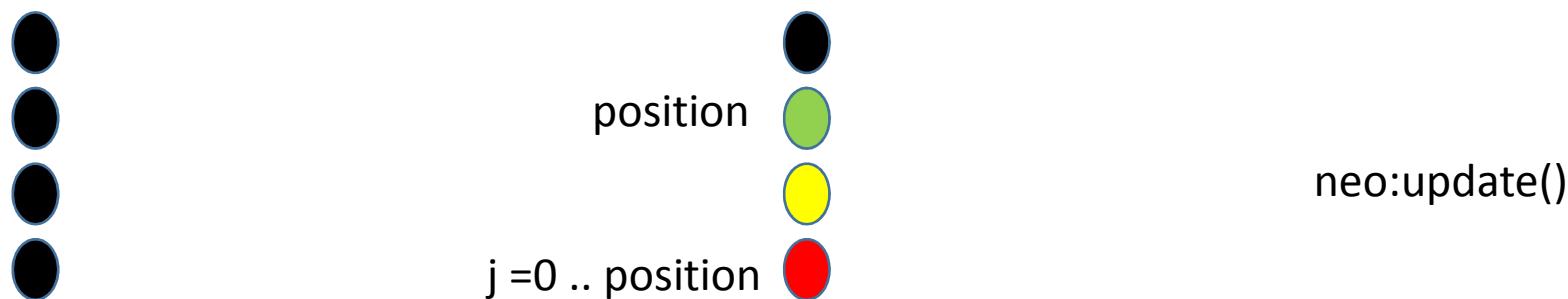
((rssи+80)/50)*7 est entre 0 et 7

Position = math.floor((rssи+80)/50)*7 est un entier entre 0 et 7

La couleur est rvb = wheelRGB((position*255) // 8)

On éteint toutes les leds

Allume en partant de 0 jqa position



| | |
|--|---|
| Initialisation neopixel | function scan_wifi() neo = neopixel.attach(neopixel.WS2812B, pio.GPIO18,8) |
| Affichage oled | while true do w = net.wf.scan(true) nb = #w; if nb > 3 then nb = 3 end cls() for j=0,nb do console(w[j].ssid.." "..w[j].rss) end |
| Calcul position | position = math.floor(7*(w[0].rss+80)/(80-30)) if position < 0 then position = 0 elseif position > 7 then position = 7 end |
| On éteint tout | for j=0,7 do neo:setPixel(j, 0,0,0) end |
| On allume en partant d'en bas jqa position | for j=0,position do r,v,b = wheelRGB((j*255)//8); neo:setPixel(j, r//10,v//10,b//10) end |
| On affiche | neo:update() |
| | tmr.delay(1) end end scan_thread = thread.start(scan_wifi) |

ESP32 + Lua RTOS

Maintenant, vous allez rendre votre montage autonome
Le script `scan_oledrgb_th.lua` doit se lancer au démarrage de l'esp32

`edit autorun.lua`

Rajouter :

```
dofile("scan_oledrgb_th.lua")
```

`CTRL-S`

`reboot`

Vérifier que le script se lance automatiquement et que votre montage est en train d'analyser le réseau WIFI

ESP32 + Lua RTOS

Remettre en l'état l'esp32 :

Stoppez le thread

```
thread.stop(scan_thread )
```

Puis enlevez la ligne dofile("scan....") de l'autorun.lua

Puis reboot

Débranchez la neopixel mais conservez l'écran oled

ESP32 + Lua RTOS

Configuration de l'ESP en mode ‘point d'accès’ (l'ESP32 fabrique son réseau Wifi)

Prenez votre téléphone et regardez la liste des points d'accès Wifi.

Le SSID **gildasmenier** doit apparaître !

ESP32 + Lua RTOS

Mini serveur Web

L'esp32 peut héberger un mini (mini) serveur Web capable de servir une dizaine maximum de clients

```
net.service.http.start()
```

```
/ > net.wf.setup(net.wf.mode.AP, "gildasmenier", "01234567")
/ > net.wf.start()
/ > net.service.http.start()
/ > http: server listening on port 80
/ > █
```

ESP32 + Lua RTOS

Branchez vous sur votre point d'accès (PC ou téléphone)

Navigateur Web

<http://192.168.4.1>

Coté ESP32 : fabriquez un fichier index.html **dans www**

<http://192.168.4.1>

```
<html>  
  <body>  
    <B> HELLO serveur </B>  
  </body>  
</html>
```

Vérifiez que vous arrivez à le lire dans un navigateur

ESP32 + Lua RTOS

Mini serveur statique de pages HTML

à partir du répertoire www

les fichiers sont envoyés tels que

NE REMPLACE PAS UN VRAI SERVEUR WEB !

Mini serveur dynamique en utilisant Lua

copier index.html dans index.lua

(utiliser la console et cp)

effacer index.html (ou le renommer)

ESP32 + Lua RTOS

Branchez vous sur votre point d'accès (PC ou téléphone)

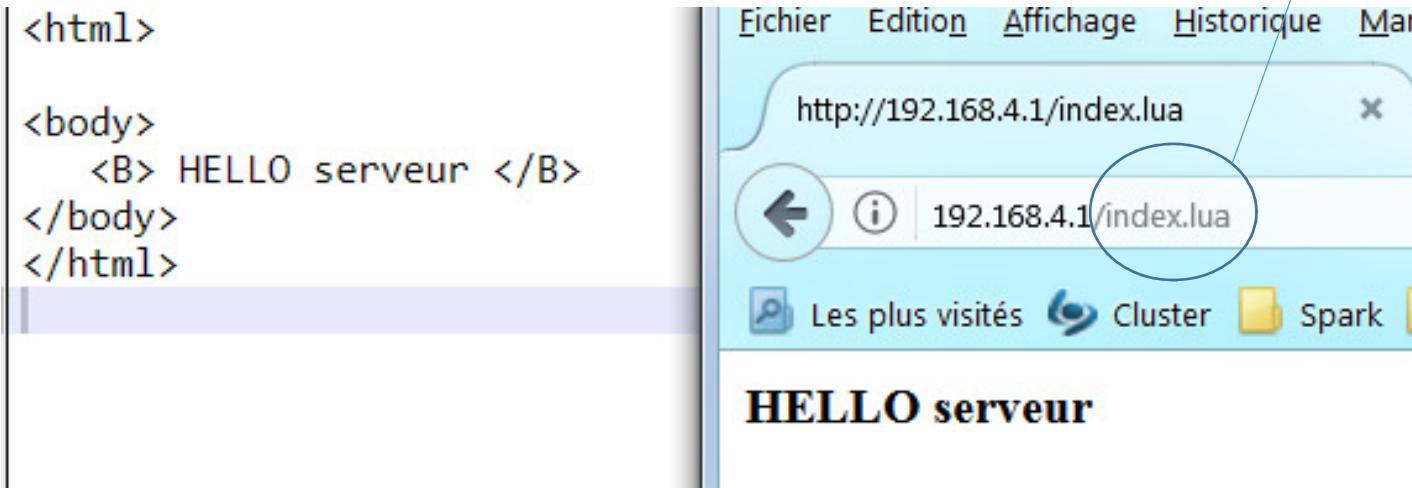
Navigateur Web

<http://192.168.4.1>

Coté ESP32 : fabriquez un fichier index.lua

<http://192.168.4.1>

facultatif



ESP32 + Lua RTOS

Preprocesseur Lua

quand l'extension du fichier est .lua
exécution des balises <?lua ?>

Essayez (*index.lua – effacez les autres fichiers de www*):

```
<h1> hello </H1>

<?lua
    console(" accès à la page index")
?>

<H1> ca va ? </H1>
```

ESP32 + Lua RTOS

On souhaite faire clignoter *une fois* la **led bleu** de l'ESP
(GPIO 02)
à chaque fois que la page d'index est demandée.

Fabriquez la page d' **index.lua**

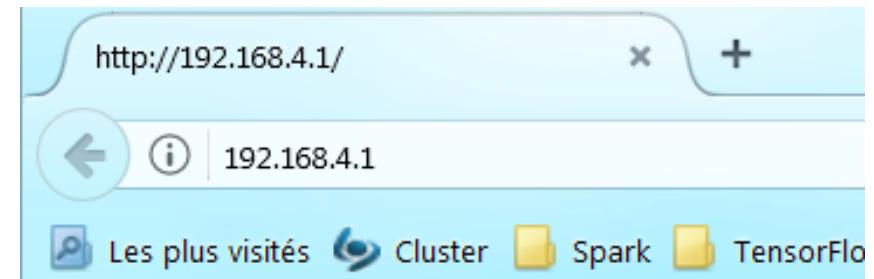
ESP32 + Lua RTOS

```
<h1> hello </H1>
```

```
<?lua
    ledon();
    tmr.delayms(200)
    ledoff();

    console("accès page")
?>
```

```
<H1> ca va ? </H1>
```



hello

resultat code Lua

ca va ?

ESP32 + Lua RTOS

On souhaite compter le nombre de chargements de la page d'index

```
<h1> hello </H1>
```

```
<?lua
```

nb = nb+1

console(nb.." chargements")

```
?>
```

```
<H1> ca va ? </H1>
```

Définition et initialisation ?

autorun.lua

```
net.wf.setup(net.wf.mode.AP, "gildasmenier",  
"01234567")  
net.wf.start()
```

nb = 0 -- variable globale

```
net.service.http.start()
```

www/index.lua



ESP32 + Lua RTOS

On souhaite compter le nombre de chargements de la page d'index

```
<h1> hello </H1>
```

```
<?lua
```

nb = nb+1

print(nb.." chargements")

```
?>
```

```
<H1> ca va ? </H1>
```

Définition et initialisation ?

autorun.lua

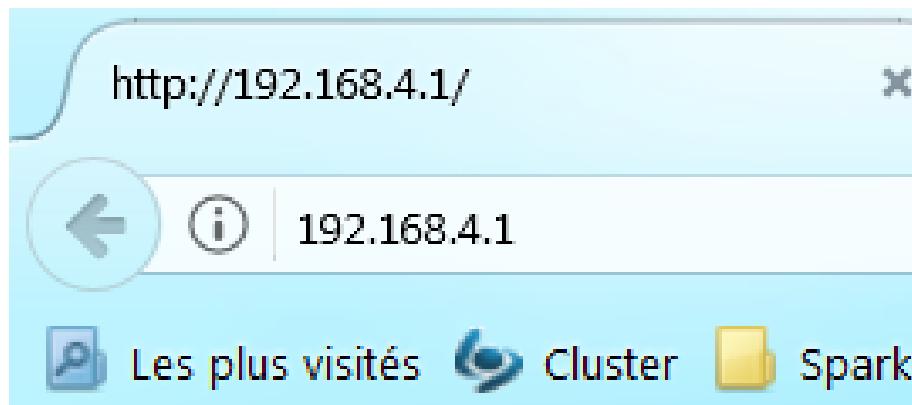
```
net.wf.setup(net.wf.mode.AP, "gildasmenier",  
"01234567")  
net.wf.start()
```

nb = 0 -- variable globale

```
net.service.http.start()
```

www/index.lua

ESP32 + Lua RTOS



hello

6 chargements

ca va ?

**print est redirigé vers la page WEB
(pas vers la sortie série)**

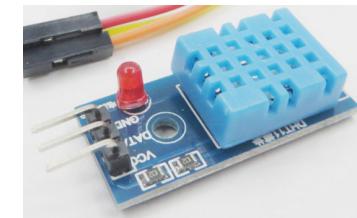
ESP32 + Lua RTOS

On souhaite maintenant interroger le capteur de température + humidité via le Wifi

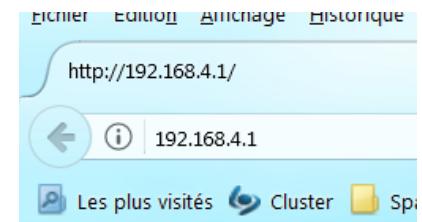
Rappel :

```
s = sensor.attach("DHT11", pio.GPIO15)
```

```
print("temp: "..s:read("temperature")..", hum: "..s:read("humidity"))
```



Modifiez index.lua et autorun.lua pour que index affiche la température et l'humidité / capteur



hello

temp: 21.0, hum: 61.0

ca va ?

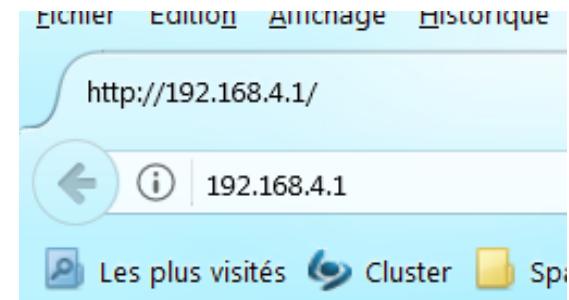
```
-- autorun.lua

net.wf.setup(net.wf.mode.AP, "bilbil", "01234567")
net.wf.start()

s = sensor.attach("DHT11", pio.GPIO15)

    -- while not net.connected() do
    -- tmr.delayms(200)
    -- end

net.service.http.start()
```



hello

temp: 21.0, hum: 61.0

ca va ?

```
-- index.lua
<h1> hello </H1>

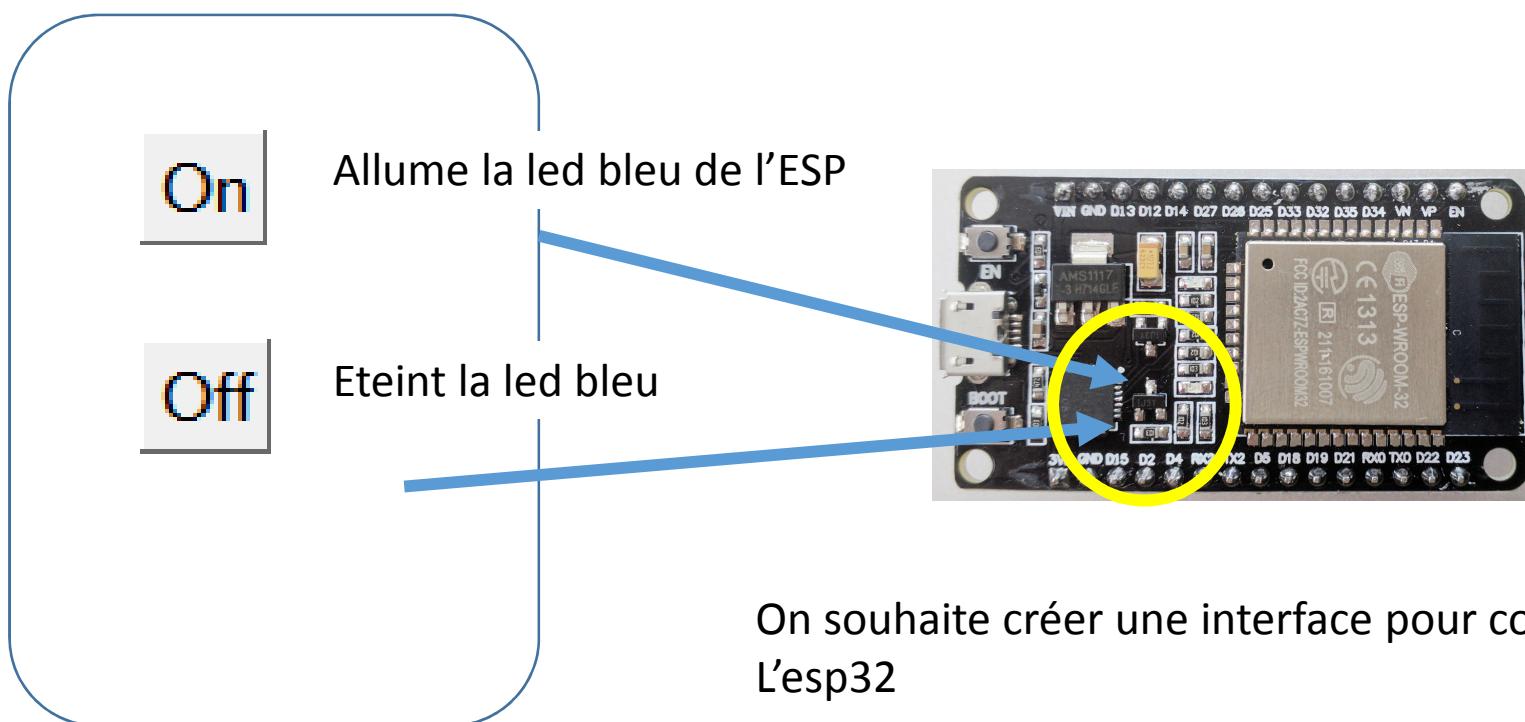
<?lua
    print("temp: "..s:read("temperature").." , hum:"..s:read("humidity"))

?>

<H1> ca va ? </H1>
```

ESP32 + Lua RTOS

On souhaite maintenant réaliser une interface de type formulaire html avec deux boutons : un pour allumer la led bleu de l'ESP et un autre pour l'éteindre.



Téléphone (fichier <http://192.168.4.1/led.html>)

ESP32 + Lua RTOS

Fabriquez 2 fichiers dans www :

ledoff.lua qui éteint la led bleu (ledoff());

ledon.lua qui allume la led bleu (ledon());

L'idée est de réaliser un formulaire HTML classique

```
<form action="ledon.lua">
  <button type="submit">On</button>
</form>
```

Création d'un bouton submit. Le click lance le chargement de ledon.lua et donc exécute ledon()

Problème : la page ledon.lua va remplacer la page led.html

ledon.lua :

```
<?lua
  ledon();
?>
```

ESP32 + Lua RTOS

```
<form action="ledon.lua" target="fd" >  
  <button type="submit">On</button>  
</form>
```

```
<form action="ledoff.lua" target="fd" >  
  <button type="submit">Off</button>  
</form>
```



invisible

led.html :

```
<html>
<body>
<iframe name="fd" style="display:none"></iframe>

<form action="ledon.lua" target="fd">
  <button type="submit">On</button>
</form>

<form action="ledoff.lua" target="fd">
  <button type="submit">Off</button>
</form>
</body>
</html>
```

autorun.lua :

```
net.wf.setup(net.wf.mode.AP, "bilbil", "01234567")
net.wf.start()

net.service.http.start()
```

ledon.lua :

```
<?lua
  ledon();
?>
```

ledoff.lua :

```
<?lua
  ledon();
?>
```

ESP32 + Lua RTOS

On peut fabriquer un serveur web accessible à partir d'un réseau internet classique :

```
net.wf.setup(net.wf.mode.STA, "reseauWIFI", "01234567")  
net.wf.start()
```

station

```
net.service.http.start()
```

Pour connaître l'adresse ip de l'esp32 :

```
net.stat()
```

```
/ > net.stat()  
wf: mac address 24:0a:c4:07:11:88  
      ip address 192.168.1.6 / netmask 255.255.255.0  
      gw address 192.168.1.254  
      ip6 address fe80:0000:0000:0000:260a:c4ff:fe07:1188  
  
en: mac address 00:00:00:00:00:00  
      ip address 0.0.0.0 netmask 0.0.0.0  
      gw address 0.0.0.0  
  
/ > |
```

Serveur ssh : dropbear

[https://matt.ucc.asn.au/
dropbear/dropbear.html](https://matt.ucc.asn.au/dropbear/dropbear.html)

- Only 1 client connection is allowed.
- Only the root user is allowed.
- The server keys are created on the fly the first time that dropbear starts.
- SCP transfers are not supported by now.
- Remote SSH command execution are not supported by now.

Dropbear SSH

Dropbear is a relatively small [SSH](#) server and client. It runs on a variety of POSIX-based platforms. Dropbear is open source software, distributed under a [MIT-style license](#). Dropbear is particularly useful for "embedded"-type Linux (or other Unix) systems, such as wireless routers.

If you want to be notified of new releases, or for general discussion of Dropbear, you can subscribe to the relatively low volume [mailing list](#).

Security Update Dropbear 2017.75 [fixes security issues](#), most users are advised to upgrade

Features

- A small memory footprint suitable for memory-constrained environments – Drop [statically linked binary](#) with uClibc on x86 (only minimal options selected)
- Dropbear server implements X11 forwarding, and authentication-agent forwarding
- Can run from inetd or standalone
- Compatible with OpenSSH `~/.ssh/authorized_keys` public key authentication
- The server, client, keygen, and key converter can be compiled into a single binary
- Features can easily be disabled when compiling to save space
- Multi-hop mode uses SSH TCP forwarding to tunnel through multiple SSH hosts in `dbclient user1@hop1,user2@hop2,destination`

Platforms

- Linux – standard distributions, [uClibc](#) >=0.9.17, [dietlibc](#), [musl libc](#), uCLinux from in
- Mac OS X (compile with PAM support)
- FreeBSD, NetBSD and OpenBSD
- Solaris – tested v8 x86 and v9 Sparc
- IRIX 6.5 (with /dev/urandom, or prngd should work)
- Tru64 5.1 (using prngd for entropy)
- AIX 4.3.3 (with gcc and Linux Affinity Toolkit), AIX 5.2 (with /dev/urandom).
- HPUX 11.00 (+prngd), TCP forwarding doesn't work
- Cygwin – tested 1.5.19 on Windows XP

ESP32 + Lua RTOS

Connexions :

I2C : électronique / capteurs / filaire

Bluetooth : périphériques proches / débranchés

Radio : distance importante (ou pas) sans infrastructure

WIFI : point d'accès

pour gérer directement un microcontrôleur

WIFI : client pour liaison internet : dimension collective

Internet des objets (IOT)

- MQTT : Message Queuing Telemetry Transport
- ISO standard
- Mécanisme publish-subscribe
- Andy Stanford-Clark (IBM) et Arlen Nipper (Cirrus Link) 1999
 - Connexion de système de surveillance de pompes à pétrole
 - Satellites
- Un des principaux protocoles de l'IOT
- MQTT V5 (2018)
- Basé sur TCP/IP

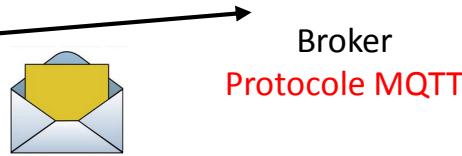
Internet des objets (IOT)

- Principe



Objet communiquant
TCP / messages MQTT

(montre, température, alerte etc..)



Le message est associé à un **SUJET** (TOPIC)



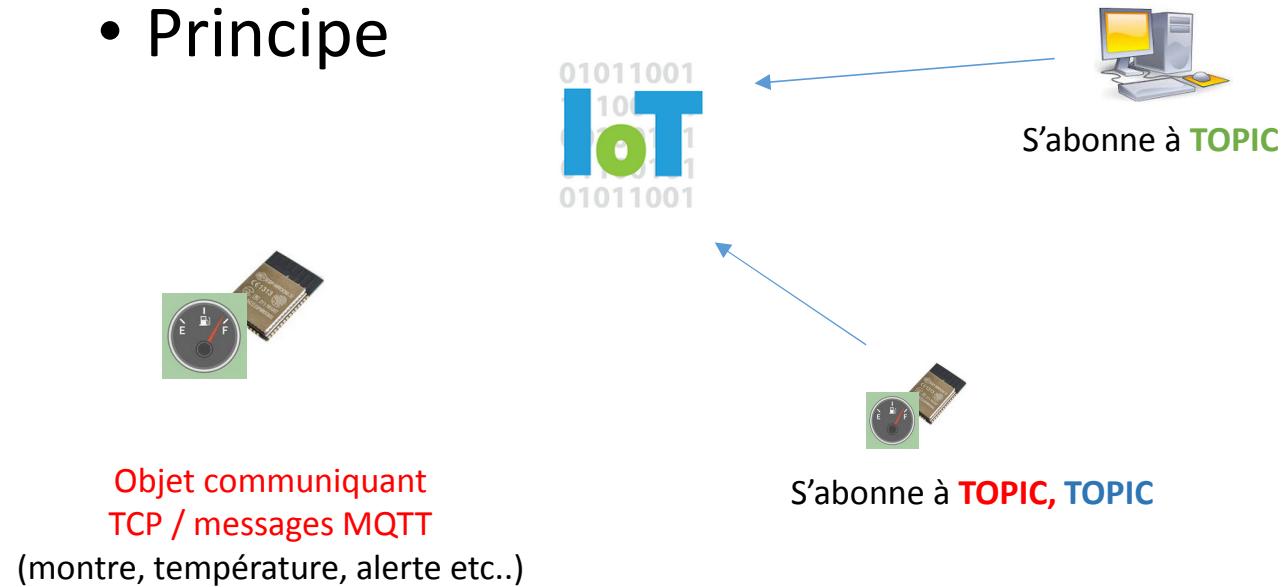
Le broker

trie les messages / sujet
gère un système d'abonnement
renvoie les messages aux
abonnés

Complètement différent de la messagerie
Pas besoin d'adresses particulières
L'objet envoie (sujet, message) et ne gère pas
la transmission d'information ensuite.

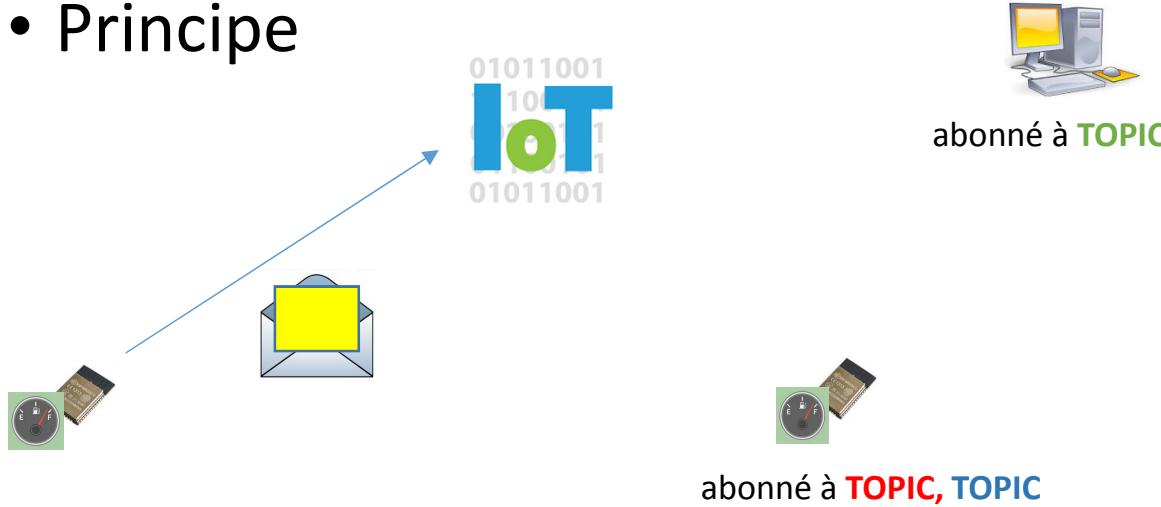
Internet des objets (IOT)

- Principe



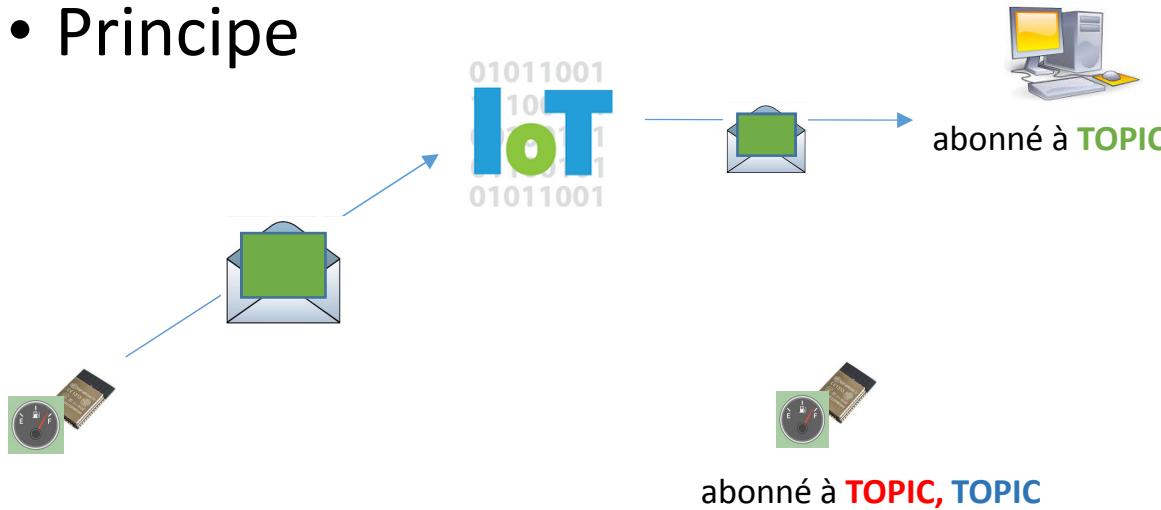
Internet des objets (IOT)

- Principe



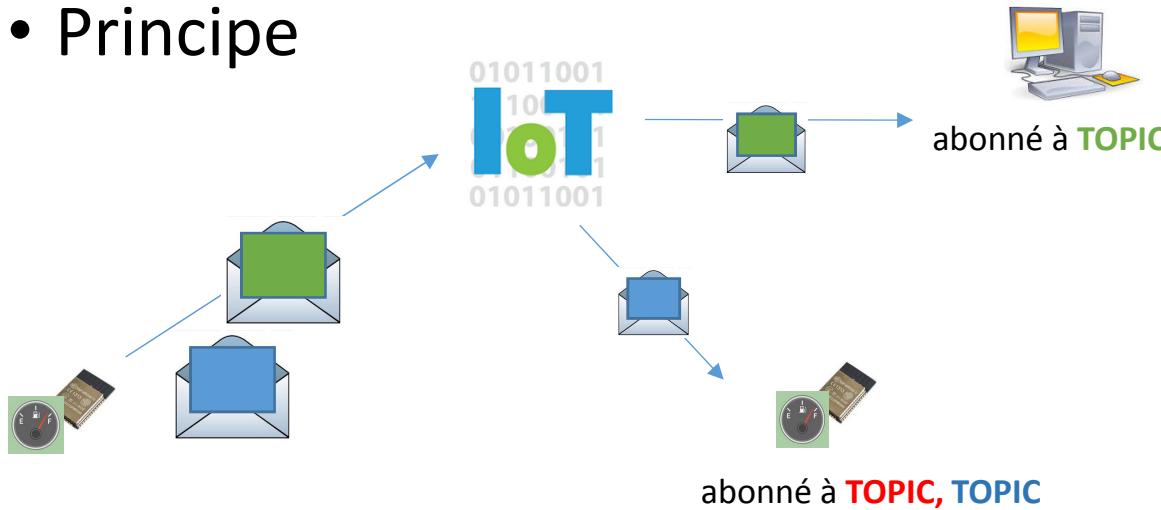
Internet des objets (IOT)

- Principe



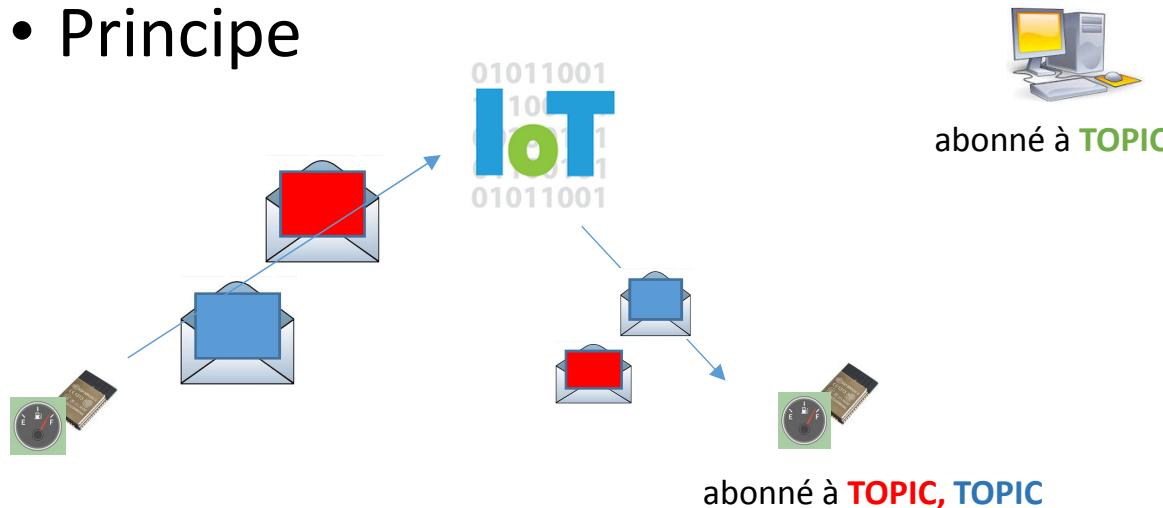
Internet des objets (IOT)

- Principe

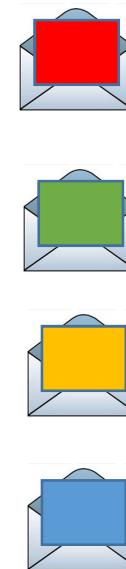


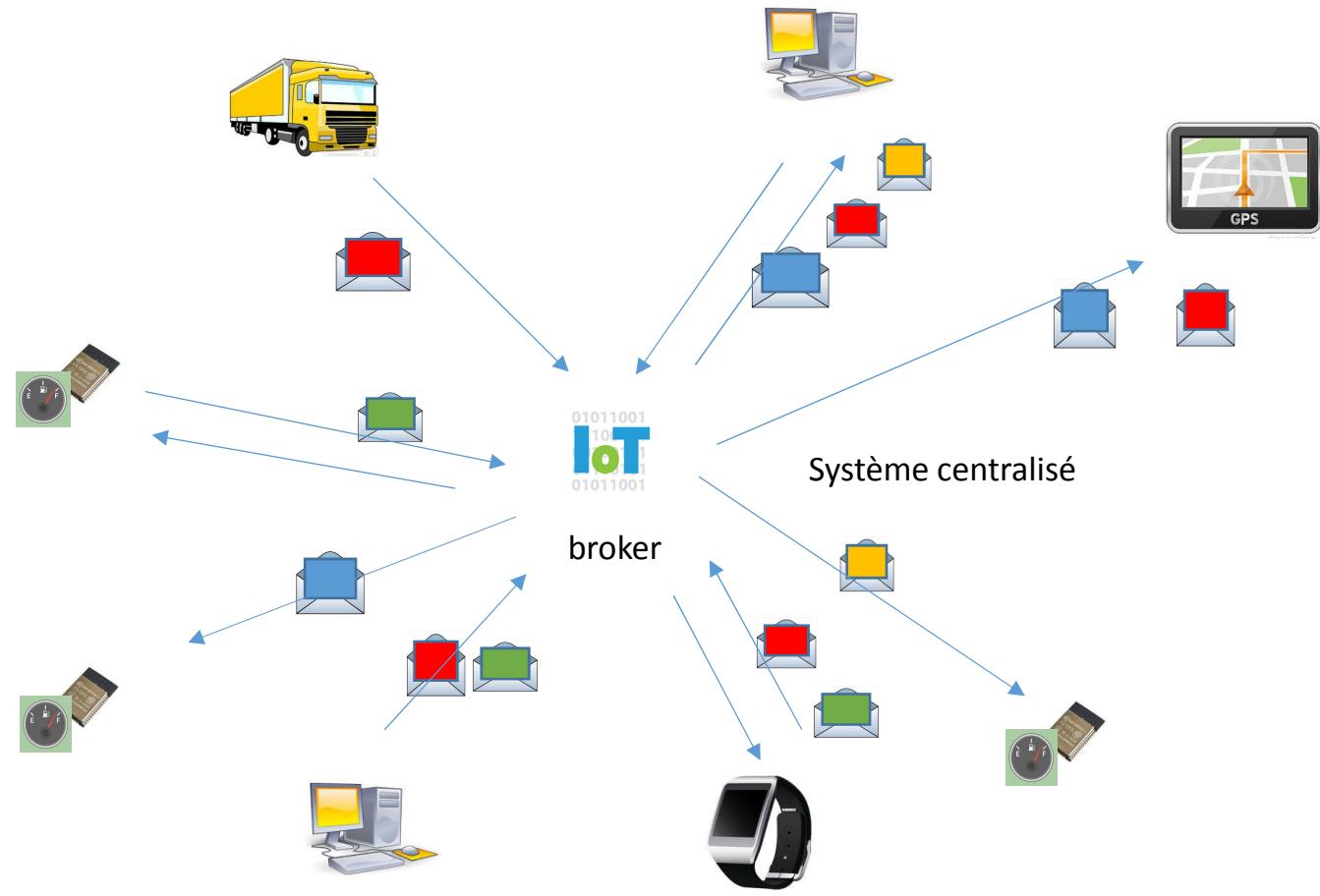
Internet des objets (IOT)

- Principe

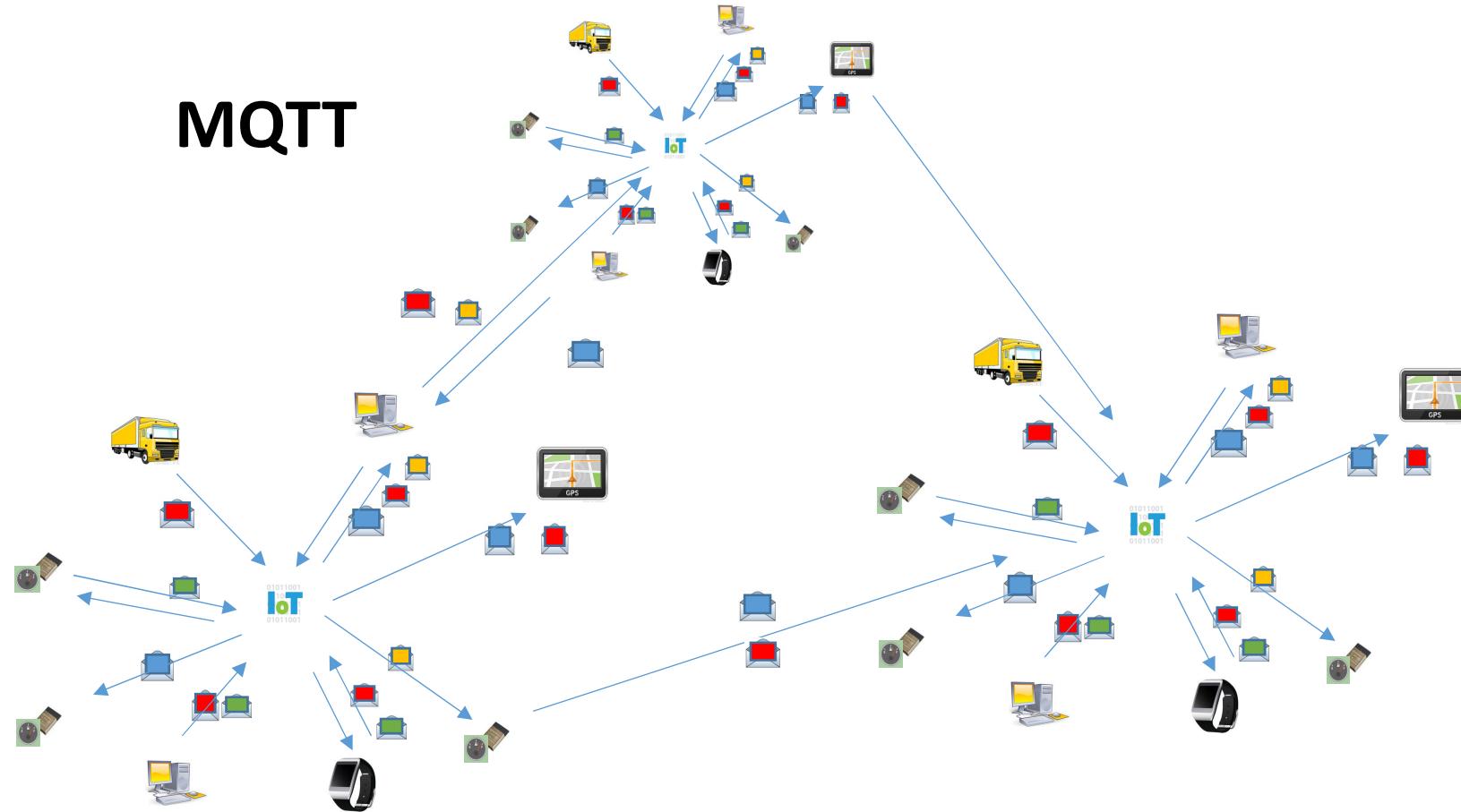


Chaque objet peut s'abonner ou se désabonner quand il veut en fonction des données locales, ou des messages reçus





MQTT



Internet des objets (IOT) : MQTT



- Mosquito

Eclipse Mosquitto™

An open source MQTT broker

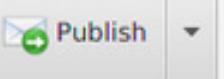
Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol suitable for use on all devices from low power single board computers to full servers.

- <https://mosquitto.org/>
 - http://mini.arsaniit.com:1883
 - Broker MQTT 3.1.1
 - Windows / Linux etc..
 - Serveur de test : test.mosquitto.org

File Configuration Connections Window Help

Control panel mqtt-spy@localhost

▼ Publish message

Topic home/office/temp
Data 21.1 

▼ Scripted publications

| Script name | Source | Repeat | Status | Messages ▾ | Last published |
|-------------|---------------|-------------------------------------|---------|------------|---------------------|
| bedroom | Script folder | <input checked="" type="checkbox"/> | Running | 224 | 2016/07/31 22:33:42 |
| office | Script folder | <input checked="" type="checkbox"/> | Running | 220 | 2016/07/31 22:33:43 |
| kitchen | Script folder | <input checked="" type="checkbox"/> | Running | 151 | 2016/07/31 22:33:37 |

▼ Subscriptions and received messages

New All home/bedroom/# home/kitchen/# home/office/# \$SYS/#

Message 1 / 1375 Show latest   

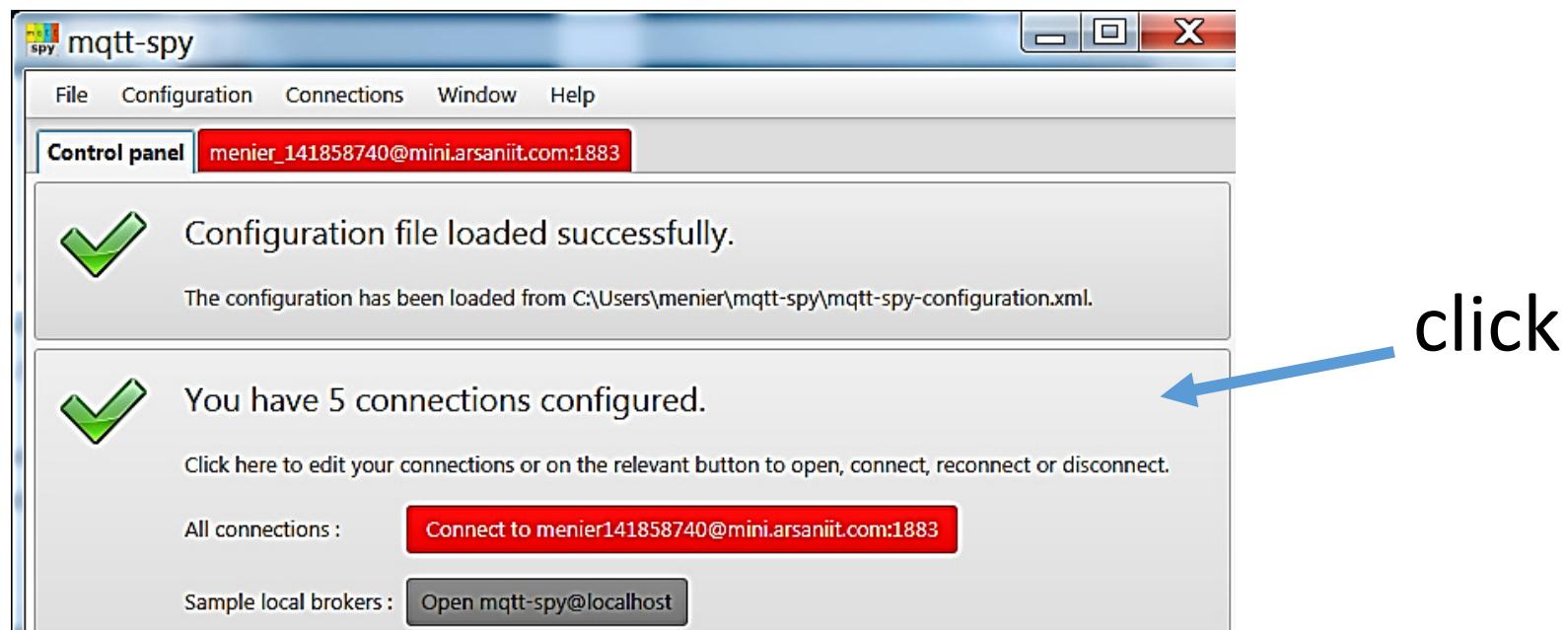
Topic home/office/current Time 2016/07/31 22:33:43:025
Data 22.1

▼ Received messages summary [search topics:] (51 topics, 1374 messages, load: 2.2/5.3/4.6)

| Topic | Content | Browse | Messages | Last received |
|----------------------|-------------------------|-------------------------------------|----------|-------------------------|
| home/office/current | 22.1 | <input checked="" type="checkbox"/> | 220 | 2016/07/31 22:33:43:025 |
| home/kitchen/current | 20.7 | <input checked="" type="checkbox"/> | 151 | 2016/07/31 22:33:37:980 |
| home/bedroom/current | 19.4 | <input checked="" type="checkbox"/> | 224 | 2016/07/31 22:33:42:517 |
| \$SYS/broker/version | mosquitto version 1.4.9 | <input checked="" type="checkbox"/> | 1 | 2016/07/31 22:28:47:001 |
| \$SYS/broker/uptime | 1441 seconds | <input checked="" type="checkbox"/> | 28 | 2016/07/31 22:33:35:510 |

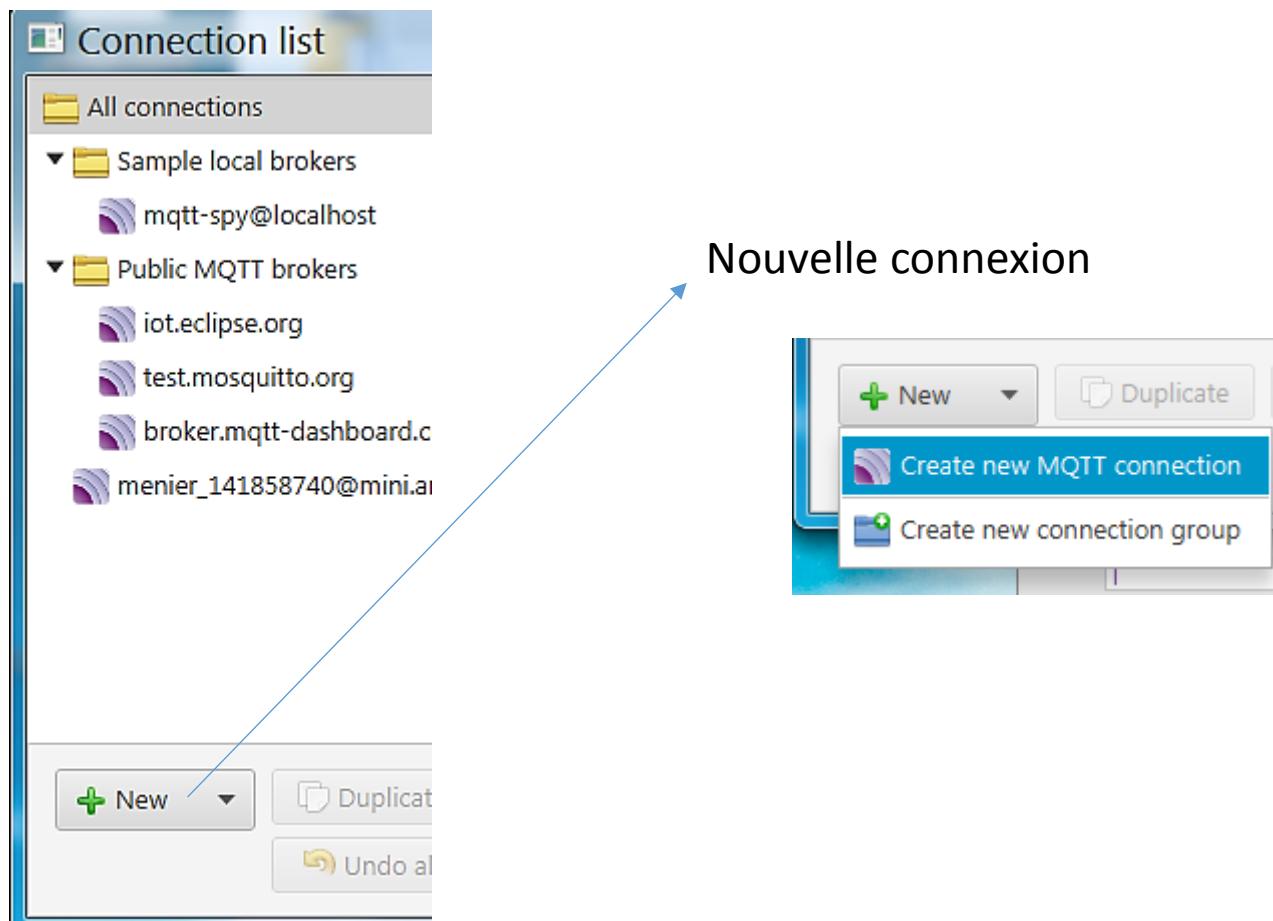
Internet des objets (IOT) : MQTT

- <https://github.com/kamilfb/mqtt-spy/wiki/Downloads>
- Utilise Java (à installer)



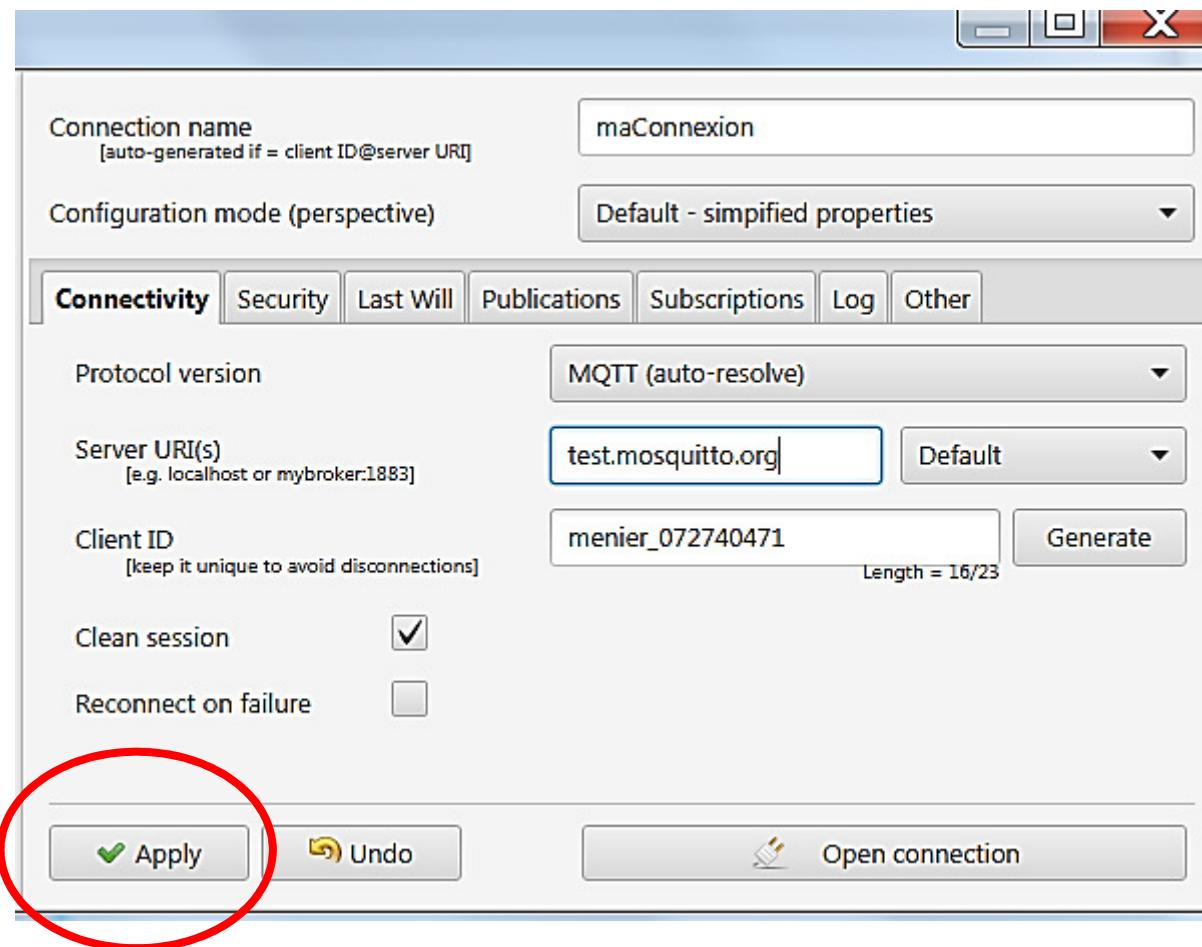
Internet des objets (IOT) : MQTT

- Création d'un nouvelle connexion cliente au broker



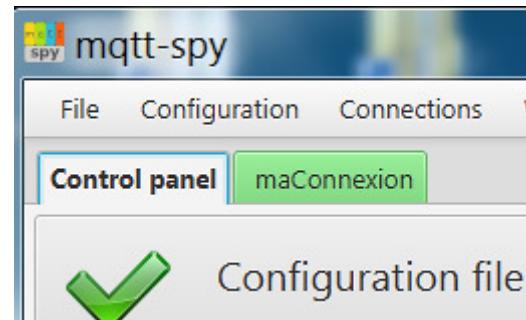
Internet des objets (IOT) : MQTT

- Création d'un nouvelle connexion cliente au broker

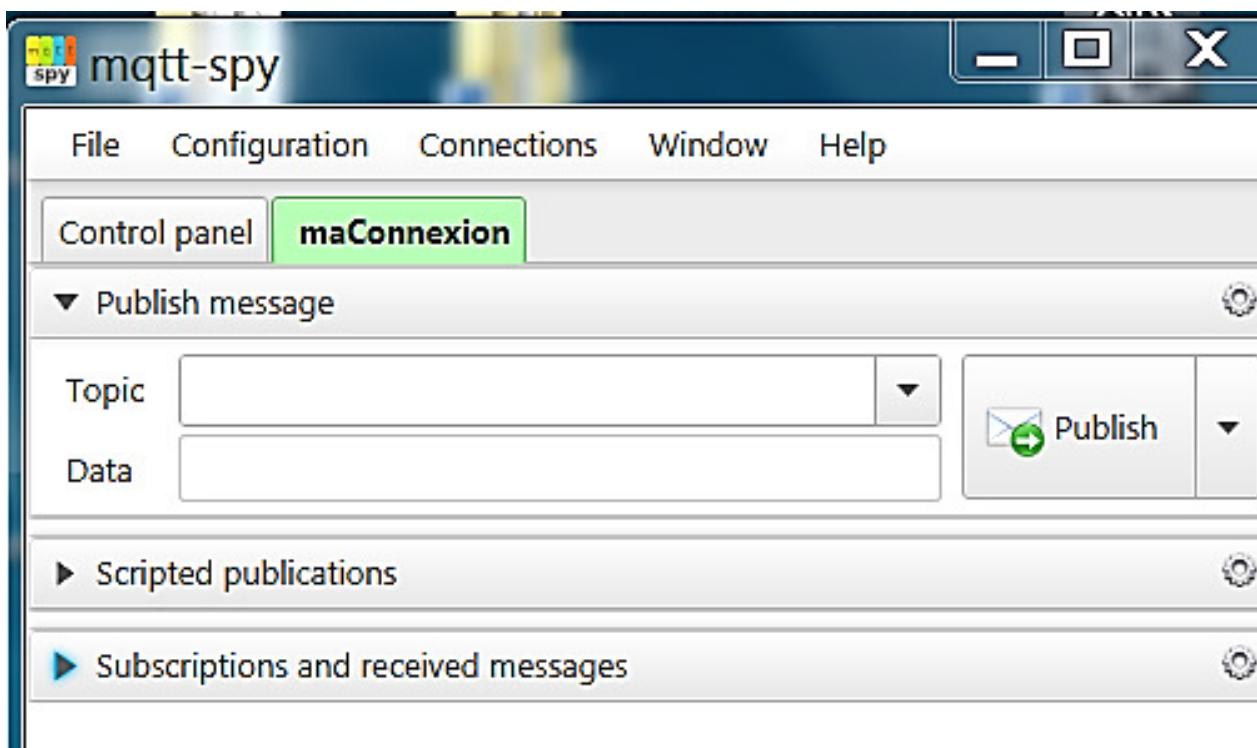


Internet des objets (IOT) : MQTT

- Open connection

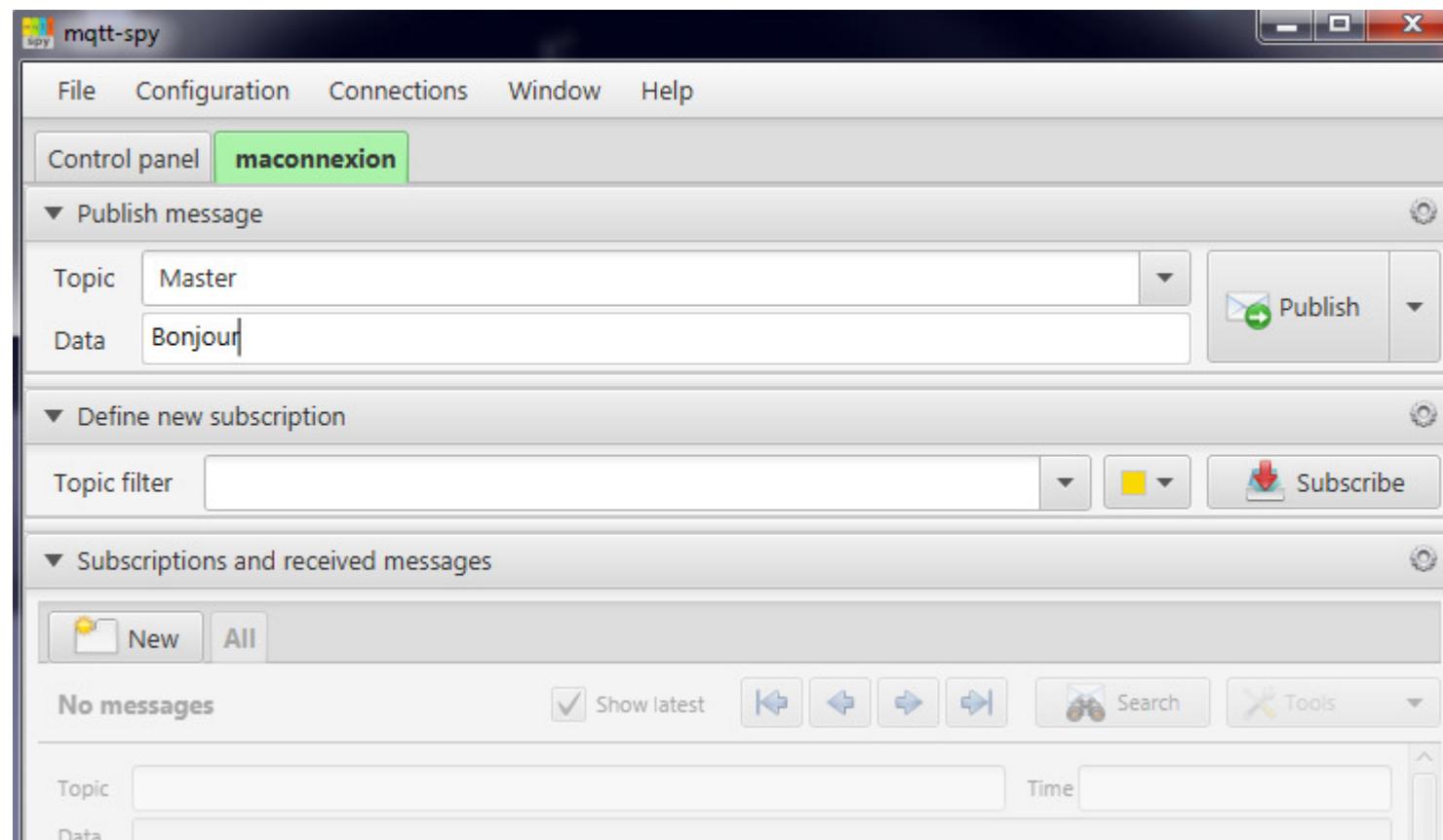


Vert = ok



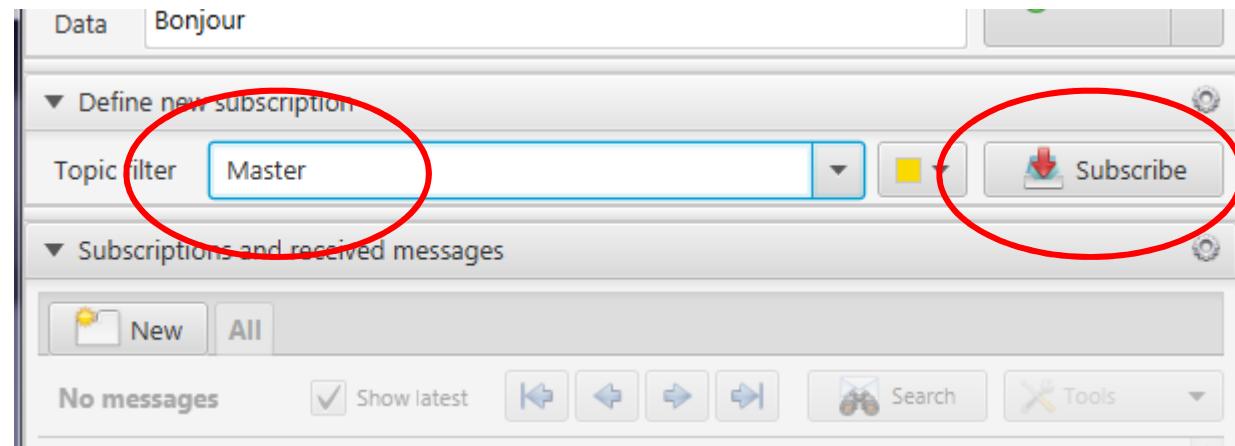
Internet des objets (IOT) : MQTT

- Topic : Master
- Message : Bonjour

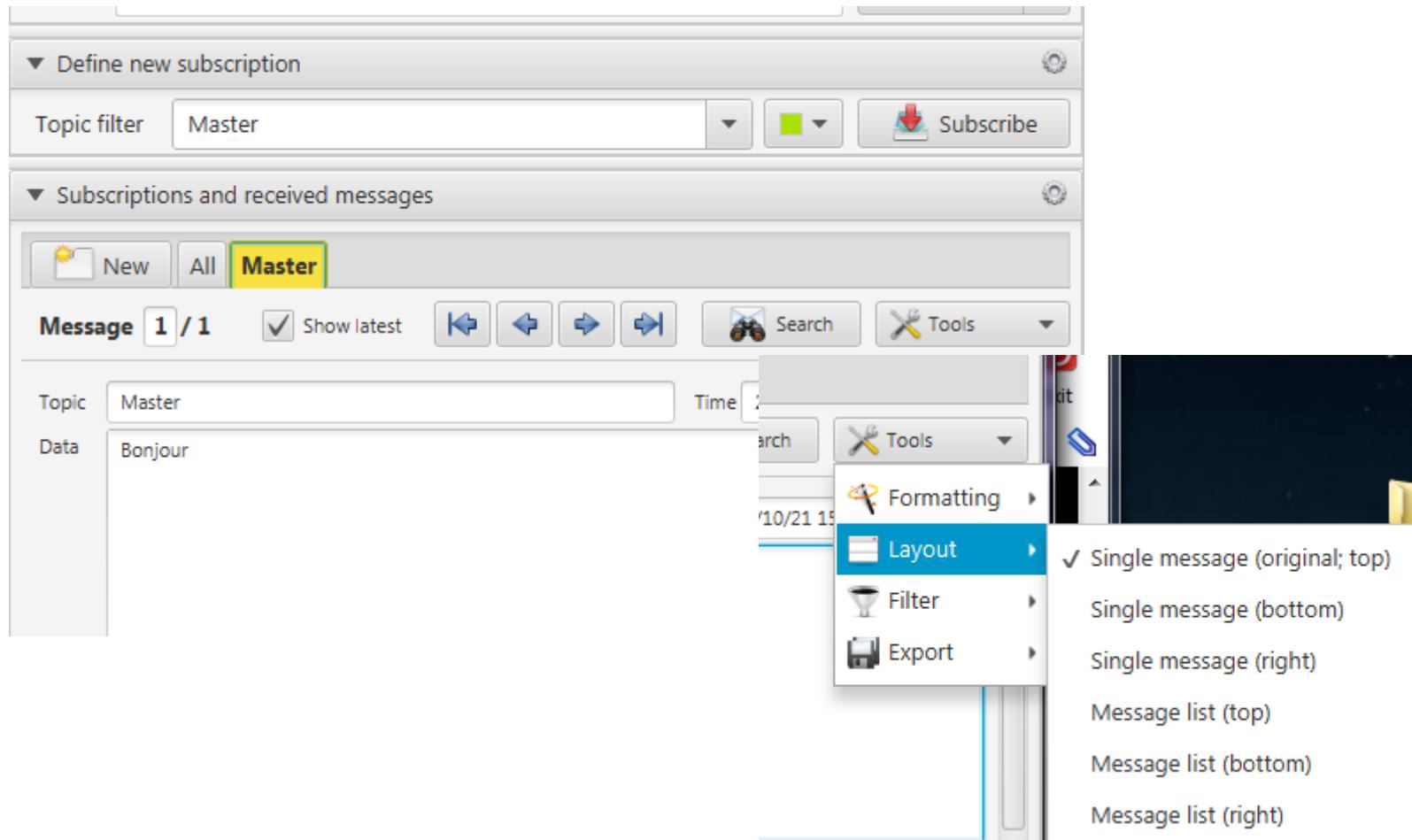


Internet des objets (IOT) : MQTT

- Inscription Topic : Master



Internet des objets (IOT) : MQTT



Internet des objets (IOT) : MQTT

- Exercices
- Refaire la même chose mais en utilisant le broker **mini.arsaniit.com (port 1883)**
- Par convention, nommage des topics :
- /topic/subtopic/subsubtopic

- Envoyez des nouvelles
- /senegal/foot
- /chanteur/....

- On travaillera désormais avec le broker mini.arsaniit.com
- (mosquitto)

MQTT et ESP32

- Les ESP32 connaissent MQTT
- 1. faire le nettoyage sur votre ESP32
 - **Vider autorun.lua**
 - **reboot**
- 2. se connecter sur un réseau WIFI
On peut utiliser `net.wf.setup(net.wf.mode.STA, "reseauWIFI", "01234567")`
`net.wf.start()`
 - **Mais on peut configurer l'esp32 pour qu'il se branche tout seul**

MQTT et ESP32

- Fichier config.lua à la racine (edit config.lua)
- Modifier :

```
-- config.wifi = true
```

Enlever (c'est un commentaire LUA)

```
-- Wifi configuration in station mode
config.data.wifi = {
    -- Put the ssid / password needed to connect to the network
    ssid = "",
    pass = "",
```

Mettre les bonnes valeurs

- Sauvegarder, reboot et vérifier avec net.stat()



Lua RTOS beta 0.1. Copyright (C) 2015 - 2018 whitecatboard.org

build 1525396213

commit d418fcb814d5d7e8743cd387df75790b913c8aed

Running from factory partition

board type ESP32THING

cpu ESP32 rev 0 at 240 Mhz

spiffs0 start address at 0x310000, size 512 Kb, partition spiffs

spiffs0 mounted

Lua RTOS beta 0.1 powered by Lua 5.3.4

Executing /system.lua ...

Starting wifi ...

i2c0 at pins scl=GPIO23/sda=GPIO22

OLED SSD1306 at i2c0

IOT - Gildas Menier - Universite de Bretagne Sud

Executing /autorun.lua ..

MQTT et ESP32

- 2. connexion au broker :

```
client = mqtt.client("master-456", "mini.arsaniit.com", 1883, false)
```

-- tmr.delay(1)

```
client:connect("", "")
```

Qualité de service

```
client:publish("MASTER", "ici l'ESP", mqtt.QOS0)
```

```
client:disconnect()
```

LE FAIRE DANS UN SCRIPT LUA (publish.lua)

MQTT et ESP32

- 2. connexion au broker :

Il faut que le nom de la connexion soit unique

```
client = mqtt.client("master-456", "mini.arsaniiit.com", 1883, false)
client.connect("", "")
client.publish("MASTER", "ici l'ESP", mqtt.QoS0)
client.disconnect()
```

- Vérifiez avec **mymqtt-spy** que vous recevez bien le message
- Comme tout le monde publie sur ce topic, réalisez votre topic de test et souscrivez sous mqtt

MQTT et ESP32

- Concrètement, un esp32 peut désormais envoyer des messages
- À d'autres esp32, MCU, ou des PC, ou des téléphones, ou d'autres microcontrôleurs qui connaissent MQTT et sont connectés sur Internet
- Sans adressage (pas besoin d'adresse IP)
- Un esp32 peut se connecter à plusieurs brokers

MQTT et ESP32

- Réception des messages
- Connexion :

```
client = mqtt.client("master-456coucou","mini.arsaniit.com", 1883, false);
client:connect("", "")
```

reception.lua

Mettre un autre nom

```
function receptionMessages(longueur, message)
    console(message)
end
```

Callback : appel quand un message est reçu

```
client:subscribe( "MASTER" , mqtt.QOS0, receptionMessages)
```

La fonction **receptionMessages** est liée au topic "**MASTER**"

Testez avec mymqtt-spy

MQTT et ESP32

- Exercices

Reception_multiple.lua

Modifiez le script pour plusieurs inscriptions
"MASTER" : affiche "MASTER : "..message

"devoir Master" : affiche "A faire: "..message

"etudiants" : affiche "promo: "..message

- Et vérifiez la réception avec myqtt-spy



MQTT et ESP32

- Exercices

- Commandez le bandeau Led RVB par MQTT
- Le message MQTT indique le numéro de la led à allumer

Lua : a = tonumber("10")

MQTT et ESP32

- Exercices

- Script avec l'encodeur
- Rotation de l'encodeur = position led allumée
- Cette fois, 2 esp 32
- Un avec le bandeau de led et l'autre avec encodeur
- Celui avec l'encodeur contrôle la led sur l'autre esp32

MQTT et ESP32

- Exercices

- Script : feu vert / orange rouge bandeau led
- Message MQTT pour lancer le feu
- Message MQTT pour stopper le feu
- Message MQTT pour passer le feu en orange clignotant

MQTT et ESP32

- Exercices

- Script : feu vert / orange rouge bandeau led
- On veut synchroniser tous les feux de la salle
 - Ils font tous la même chose
 - Ils changent en même temps mais n'ont pas forcément la même couleur
- Stratégie ?
 - Centralisée ?
 - Gestion de panne ?

MQTT et ESP32

alerte_banque.lua
alerte_police.lua

- Application
 - Chaque porte de la banque est équipée d'un verrou automatique
 - Quand une alerte a lieu, toutes les portes de la banque se ferment
 - Les portes restent fermée jusqu'à la fin de l'alerte
 - La police est alertée en cas d'alerte : c'est la police qui décide de la fin de l'alerte (après inspection).
- Analyse IOT :
 - Un microcontrôleur par porte avec batteries intégrées
 - Chaque microcontrôleur reçoit des informations topic **signal**
 - Quand le message est « **alerte** » le verrou se ferme
 - Quand le message est « **fin alerte** » le verrou s'ouvre
 - Toutes les serrures sont indépendantes
 - La police est également *client* du *broker* et reçoit les messages

MQTT et ESP32

alerte_banque.lua
alerte_police.lua

- Verrou automatique
 - Servo moteur
 - GND, + et contrôle
 - En envoyant un signal sur contrôle, il est possible de faire tourner le moteur à une position angulaire relativement précise.

Ici : entre 0 et 180 degrés



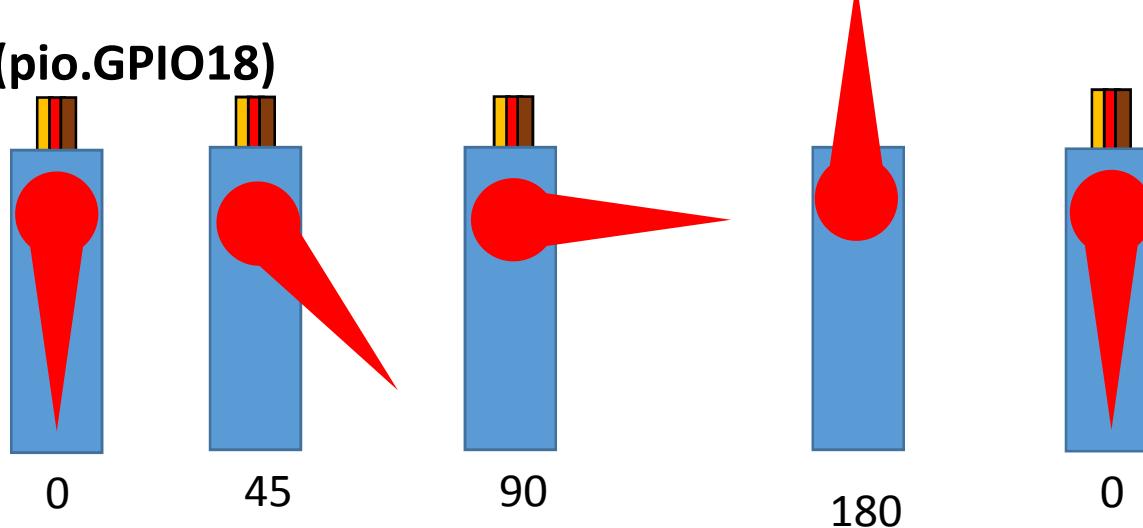
MQTT et ESP32

- Branchement ESP32

- Fil noir : GND
- Fil rouge : 3.3V
- Fil jaune : D18



- `s = servo.attach(pio.GPIO18)`
- `s.write(0)`
- `s.write(45)`
- `s.write(90)`
- `s.write(180)`
- `s.write(0)`
- `s.detach()`



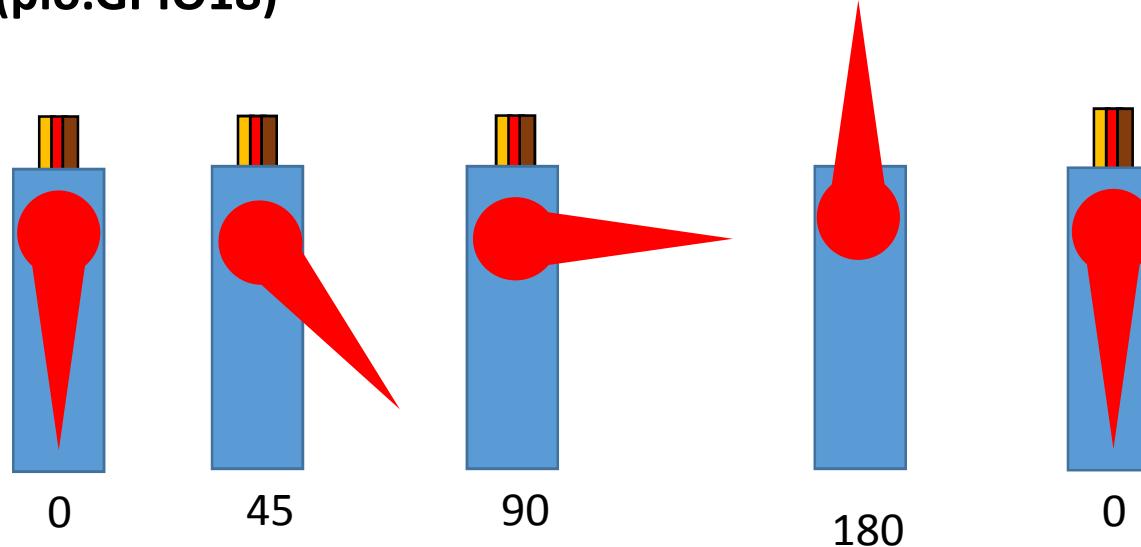
MQTT et ESP32

- Branchement ESP32

- Testez le moteur et vissez l'axe



- `s = servo.attach(pio.GPIO18)`
 - `s.write(0)`
 - `s.write(45)`
 - `s.write(90)`
 - `s.write(180)`
 - `s.write(0)`
 - `s.detach()`



MQTT et ESP32

alerte_banque.lua

```
client = mqtt.client("master-menier","ymir", 1883, false);

client:connect("", "")

function reception_signal(longueur, message)
  if message== "alerte" then
    s = servo.attach(pio.GPIO18)
    s:write(180)
    s:detach()
  elseif message == "fin alerte" then
    s = servo.attach(pio.GPIO18)
    s:write(0)
    s:detach()
  end
end

client:subscribe( "signal" , mqtt.QOS0, reception_signal)
```

MQTT et ESP32

alerte_banque.lua

```
client = mqtt.client("master-menier","ymir", 1883, false);
client:connect("", "")

function reception_signal(longueur, message)
    if message=="alerte" then
        s = servo.attach(pio.GPIO18)
        s:write(180)
        tmr.delay(2)
        s:detach()
    elseif message == "fin alerte" then
        s = servo.attach(pio.GPIO18)
        s:write(0)
        tmr.delay(2)
        s:detach()
    end
end
client:subscribe( "signal" , mqtt.QOS0, reception_signal)
```

Il faut laisser le temps au moteur
pour qu'il atteigne sa position !

MQTT et ESP32

alerte_police.lua

```
client = mqtt.client("master-menier","ymir", 1883, false);
client:connect("", "")

function reception_signal(longueur, message)
    if message=="alerte" then
        ledon()
    elseif message == "fin alerte" then
        ledoff()
    end
end
client:subscribe( "signal" , mqtt.QOS0, reception_signal)
```

Exercice

publish_temp_th.lua

- Réalisez un montage capable de transmettre la température et le degré d'humidité toutes les 10s sur le topic
/ISGA/mesures/temp
- Comme on souhaite pouvoir savoir qui expédie le message, votre message aura la forme :

votre_nom : 24

Rappel :

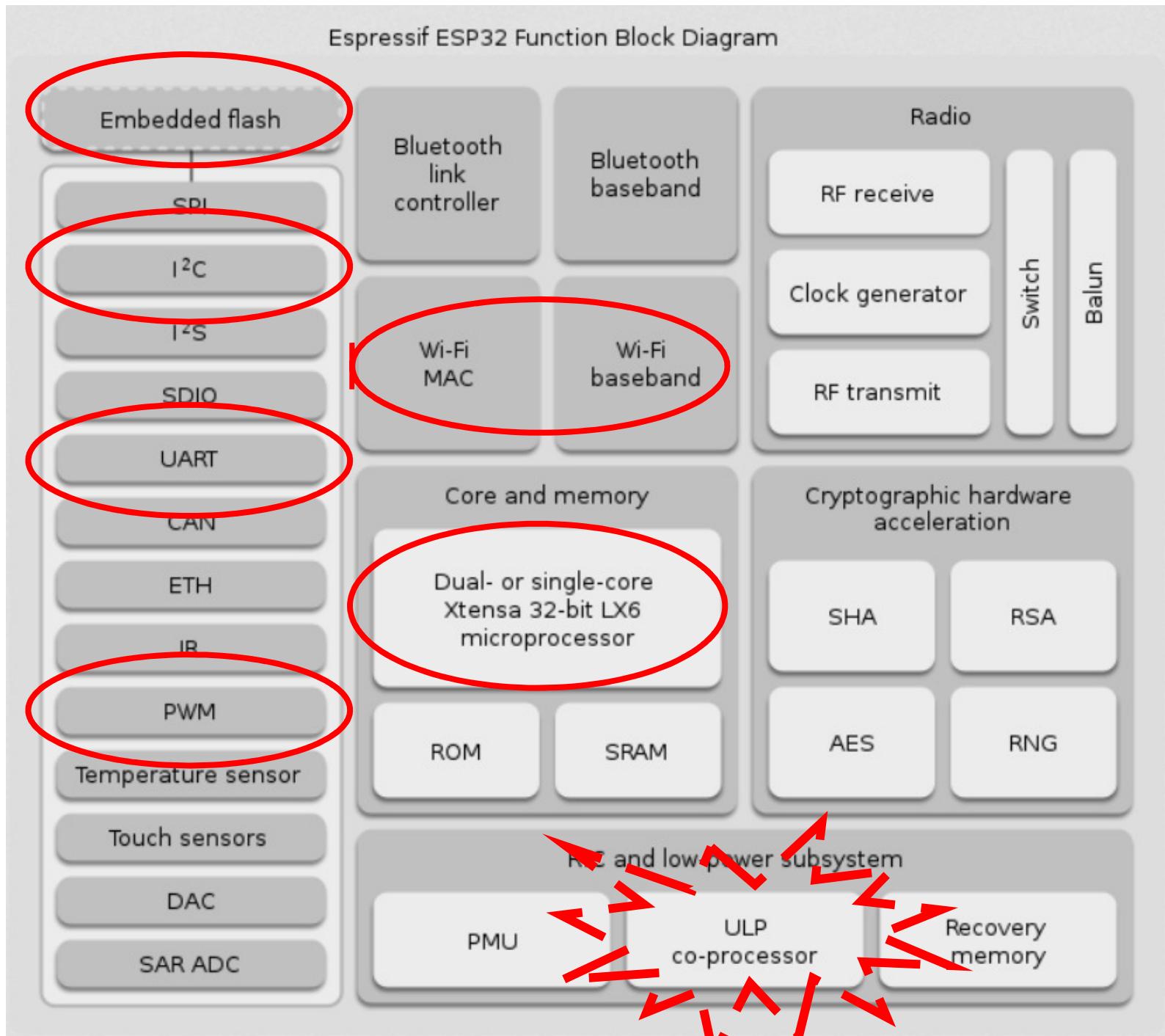
```
s = sensor.attach("DHT11", pio.GPIO15)
```

```
while true do
    print("temp: ..s:read("temperature").." humidite : "..s:read("humidity"))
    tmr.delay(10)
end
```

```
"menier".. " : ..temperature
```



DHT11 en D15



Attention !

publish_temp_th.lua

- Vous allez utiliser une technique différente des threads
- Très différente de l'utilisation des PCs
- **Typique de l'IOT**
- **Économie d'énergie**
- **Principe**
 - L'EP32 se lance
 - Se branche sur le réseau WIFI local
 - Fait sa (ou ses) mesures
 - Expédie son message sur le broker et son topic
 - Passe en mode sommeil pendant un certain temps : **os.sleep(10)**
 - L'ESP32 ne consomme que très très peu de courant
 - Pas de connexion WIFI
 - Horloge interne
 - Au bout d'un certain temps (10s), l'esp32 se rebooté
 - On recommence !

```
ledon()
s = sensor.attach("DHT11", pio.GPIO15)
```

publish_sleep.lua

```
nom = "menier"
```

```
temp = s:read("temperature")
```

```
client = mqtt.client("master_menier", "ymir", 1883, false)
client:connect("", "")
client:publish("/UBS/mesures/temp", nom.." ..temp", mqtt.QOS0)
client:disconnect()
```

```
ledoff()
```

os.sleep(10)

mettre dofile("publish_sleep.lua") dans autorun.lua

Gestion d'exception

Si le broker n'est pas accessible ?

Si la liaison Wifi n'est pas disponible ?

S'il y a un problème avec le capteur ?

Arrêt du script (!)

Équivalent de try/catch en Lua

Plus simple : pcall (protected call)

pcall(fonction) : vrai si executé sans problème, faux si exception

`pcall(function()`

`...`

`...`

`end`

`)`

Protège l'extérieur des exceptions du contenu

```
ledon()
s = sensor.attach("DHT11", pio.GPIO15)

nom = "menier"
temp = s:read("temperature")
client = mqtt.client("master_menier", "ymir", 1883, false)

pcall(function() client:connect("", "") end)

tmr.delayms(50) end
client:publish("/MASTER/mesures/temp", nom.."": "..temp , mqtt.QOS0)
client:disconnect()
ledoff()

os.sleep(10)
```

publish_sleep.lua

```
ledon()

s = sensor.attach("DHT11", pio.GPIO15)
nom = "menier"

pcall( function()
      temp = s:read("temperature")
      client = mqtt.client("master_menier", "ymir", 1883, false)

      client:connect("", "")
      client:publish("/MASTER/mesures/temp", nom.." : ..temp , mqtt.QOS0)
      client:disconnect()
    end

ledoff(); net.wf.stop()
os.sleep(10)
```

La partie en rouge ne peut pas stopper le script même si une exception est levée

mettre dofile("publish_sleep.lua") dans autorun.lua

Internet des objets

- Comment stopper reboot / redémarrage ?



```
Lua RTOS beta 0.1. Copyright (C) 2015 - 2018 whitecatboard.org
```

```
build 1525396213
```

```
commit d418fc814d5d7e8743cd387df75790b913c8aed
```

```
Running from factory partition
```

```
board type ESP32THING
```

```
cpu ESP32 rev 0 at 240 Mhz
```

```
spiffs0 start address at 0x310000, size 512 Kb, partition spiffs
```

```
spiffs0 mounted
```

```
Lua RTOS beta 0.1 powered by Lua 5.3.4
```

```
Lua RTOS-boot-scripts-aborted-ESP32
```

```
/> █
```

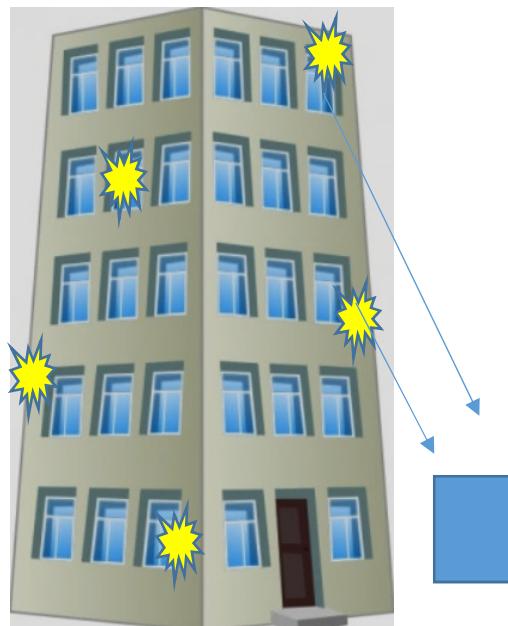
- **CTRL-D** et vider autorun.lua si nécessaire

Application

Mettre en place un réseau de capteurs de surveillance de températures dans un bâtiment : chaque capteur est un objet connecté.

Le chef de surveillance veut avoir un système qui lui donne sans arrêt :

- la pièce la plus froide (et sa température)
- la pièce la plus chaude (et sa température)



Vous avez déjà les systèmes de capteurs connectés (!)

Reste maintenant à récupérer les données à les traiter et à les rendre disponibles.

Messages : "menier : 24"

Traitement de chaines lua

```
ch = "menier : 24.0"
```

```
debut,fin = ch:find(" : ")
```

debut = 7, fin = 9

m e n i e r _ : _ 2 4 . 0
1 2 3 4 5 6 7 8 9 10

```
ch.sub(0,debut-1)      "menier"
```

```
ch.sub(fin+1)          "24.0"
```

```
tonumber(ch.sub(fin+1))
```

Reception mqtt

```
client = mqtt.client("master-chef", "ymir", 1883, false);
tmr.delay(1)
client:connect("", "")

temp_max = -1
lieu_max = "?"

temp_min = 200
lieu_min = "?"

function receptionTemp(longueur, ch)
    debut, fin = ch:find(" : ")
    nom = ch:sub(0, debut-1)
    temp= tonumber(ch:sub(fin+1))
    if temp > temp_max then temp_max = temp; lieu_max = nom end
    if temp < temp_min then temp_min = temp; lieu_min = nom end
end

client:subscribe( "/MASTER/mesures/temp" , mqtt.QOS0, receptionTemp)
```

Reception mqtt

```
function receptionTemp(longueur, ch)
    debut, fin = ch:find(" : ")
    nom = ch:sub(0, debut-1)
    temp= tonumber(ch:sub(fin+1))
    if temp > temp_max then temp_max = temp; lieu_max = nom end
    if temp < temp_min then temp_min = temp; lieu_min = nom end
    print("max : ..lieu_max.." : ..temp_max)
    print("min : ..lieu_min.." : ..temp_min)
    print("")
```

end

Pack

LuaRTOS contient une fonction très pratique pour l'émission de messages esp32 -> esp32

```
v = pack.pack(1,"hello",45,67.4,"ca va ?")
```

pack.pack(nombre d'arguments variables)

Convertit la liste des arguments en chaîne de caractères :

```
/ > v = pack.pack(1,"hello",45,67.4,"ca va ?")
/ > v
051410400100000068656C6C6F002D000000CDCC86426361207661203F00
/ > █
```

On peut envoyer cette chaîne dans un Topic

Pack

Décodage (réception) :

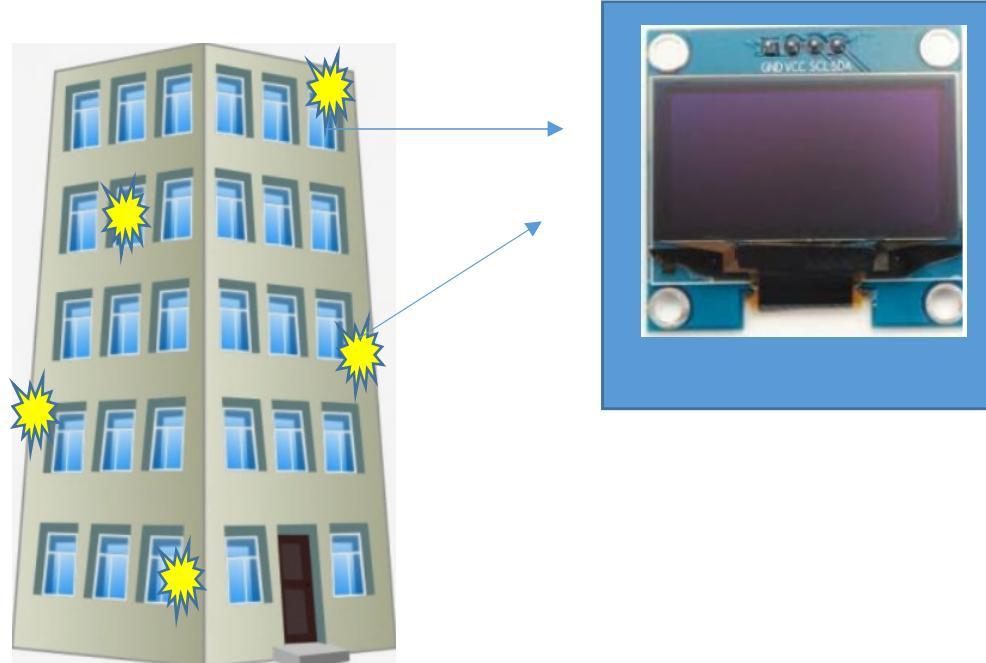
```
a, b, c, d, e = pack.unpack(v)
```

Le déconstructeur récupère les différentes parties à partir de la Chaîne.

```
/ > a,b,c,d,e = pack.unpack(v)
/ > a
1
/ > b
hello
/ > c
45
/ > d
67.4
```

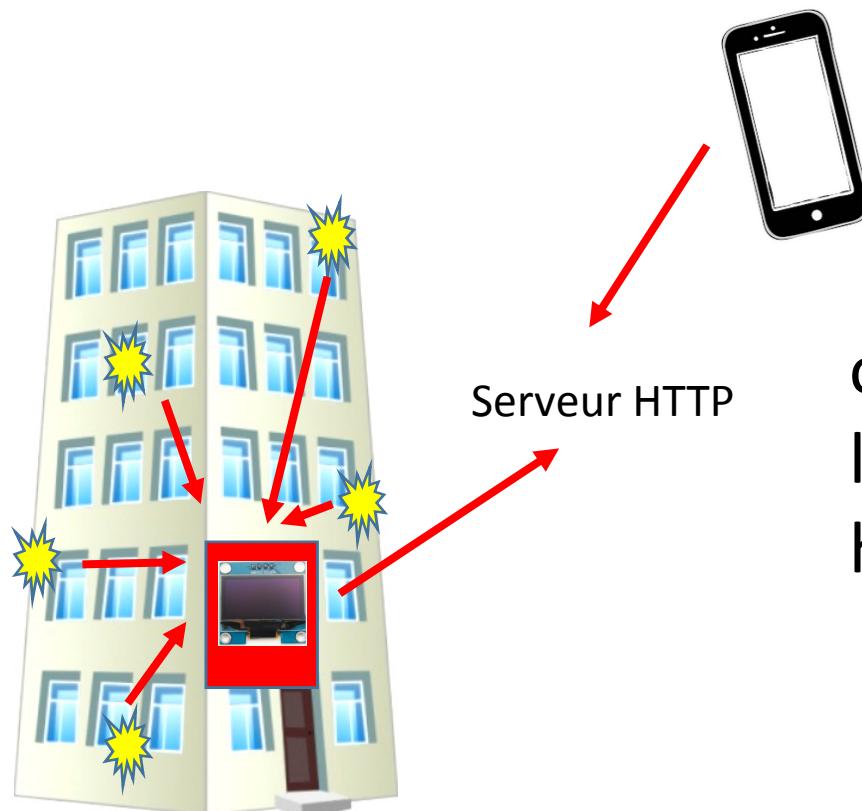
Application

Vous pouvez fabriquer un appareil avec un écran oled qui se lance dès le branchement et qui affiche régulièrement sur l'écran les informations demandées.



Application

On souhaite que le responsable ait accès aux informations à partir de son téléphone :



on va donc configurer
l'esp32 en serveur
http

Rappel

Une fois l'esp32 branché sur un réseau WIFI,

On peut connaître son adresse ip : **net.stat(true)[0]["ip"]**

on peut lancer le serveur http par : **net.service.http.start()**

Il sera accessible à l'aide d'un navigateur à l'adresse donnée par **net.stat(true)[0]["ip"]**

Par exemple **10.0.0.1** :

Navigateur téléphone : **http://10.0.0.1**

```
print("http://"..net.stat(true)[0]["ip"])
net.service.http.start()
```

batiment_http.lua

```
client = mqtt.client("master-chef","ymir", 1883, false);
tmr.delay(1)
client:connect("", "")
```

```
temp_max = -1 ; lieu_max = "?"
temp_min = 200 ; lieu_min = "?"
```

```
function receptionTemp(longueur, ch)
debut, fin = ch:find(" : ")
nom = ch:sub(0, debut-1)
temp= tonumber(ch:sub(fin+1))
if temp > temp_max then temp_max = temp; lieu_max = nom end
if temp < temp_min then temp_min = temp; lieu_min = nom end
print("max : "..lieu_max.." : "..temp_max)
print("min : "..lieu_min.." : "..temp_min)
print("")
```

```
end
```

```
client:subscribe( "/ISGA/mesures/temp" , mqtt.QOS0, receptionTemp)
```

Rappel

Reste à réaliser une page **index.lua** dans www

```
<html>
<body>
    <H1> Systeme de surveillance temp </H1>
    <?lua
        print("max = "..lieu_max.." : "..temp_max)
    ?>
    <br></br>
    <?lua
        print("min = "..lieu_min.." : "..temp_min)
    ?>
</body>
</html>
```

Remarques

Il est possible de fixer l'adresse IP du serveur

Il est également possible de rajouter un identifiant DNS

L'esp32 peut fonctionner en mode serveur SSH
on peut se connecter à distance à l'esp32
pas besoin de liaison série à partir du moment où il est
branché WIFI (par exemple)

Cryptage matériel intégré (sécurité)

MQTT et ordinateur : javascript et node.js

npm : mqtt à installer

```
const mqtt = require('mqtt')
const client = mqtt.connect('mqtt://ymir')
```

```
client.on('connect', () => {
    client.publish('banque/signal', 'alerte')
})
```

```
client.on('message', (topic, message) => {
    if (topic === "ISGA") {
        .....
    }
})
```

JAVA

```
1 import org.eclipse.paho.client.mqttv3.MqttCallback;           /**
2 import org.eclipse.paho.client.mqttv3.MqttClient;             *
3 import org.eclipse.paho.client.mqttv3.MqttConnectOptions      * deliveryComplete
4 import org.eclipse.paho.client.mqttv3.MqttDeliveryToken;       * This callback is invoked when a message published by this client
5 import org.eclipse.paho.client.mqttv3.MqttException;          * is successfully received by the broker.
6 import org.eclipse.paho.client.mqttv3.MqttMessage;            *
7 import org.eclipse.paho.client.mqttv3.MqttTopic;              */
8
9 public class SimpleMqttClient implements MqttCallback {
10
11     MqttClient myClient;
12
13     MqttConnectOptions connOpt;                                /**
14
15     static final String BROKER_URL = "tcp://q.m2m.io";        *
16     static final String M2MIO_DOMAIN = "<Insert m2m.";        * messageArrived
17     static final String M2MIO_STUFF = "things";                * This callback is invoked when a message is received on a subscribed topic
18     static final String M2MIO_THING = "<Unique devic";        *
19     static final String M2MIO_USERNAME = "<m2m.io us";        */
20     static final String M2MIO_PASSWORD_MD5 = "<m2m.i @Override
21
22 // the following two flags control whether this
23 static final Boolean subscriber = true;                      public void messageArrived(MqttTopic topic, MqttMessage message) throws Ex
24 static final Boolean publisher = true;                        System.out.println("-----")
25
26 /**
27 * connectionLost                                         */
28 * This callback is invoked upon losing the MQTT      * MAIN
29 *                                                       *
30 */                                                       */
31 @Override
32 public void connectionLost(Throwable t) {                  public static void main(String[] args) {
33     System.out.println("Connection lost!");               SimpleMqttClient smc = new SimpleMqttClient();
34     // code to reconnect to the broker would }           smc.runClient();
35 }
```

JAVA

JAVA

```
/*
 *
 * runClient
 * The main functionality of this simple example.
 * Create a MQTT client, connect to broker, pub/sub, disconnect.
 *
 */
public void runClient() {
    // setup MQTT Client
    String clientID = M2MIO_THING;
    connOpt = new MqttConnectOptions();

    connOpt.setCleanSession(true);
    connOpt.setKeepAliveInterval(30);
    connOpt.setUserName(M2MIO_USERNAME);
    connOpt.setPassword(M2MIO_PASSWORD_MD5.toCharArray());

    // Connect to Broker
    try {
        myClient = new MqttClient(BROKER_URL, clientID);
        myClient.setCallback(this);
        myClient.connect(connOpt);
    } catch (MqttException e) {
        e.printStackTrace();
        System.exit(-1);
    }

    System.out.println("Connected to " + BROKER_URL);

    // setup topic
    // topics on m2m.io are in the form <domain>/<stuff>/<thing>
    String myTopic = M2MIO_DOMAIN + "/" + M2MIO_STUFF + "/" +
    MqttTopic topic = myClient.getTopic(myTopic);

    // subscribe to topic if subscriber
    if (subscriber) {
        try {
            int subQoS = 0;
            myClient.subscribe(myTopic, subQoS);
        } catch (Exception e) {
        }
    }
}

// publish messages if publisher
if (publisher) {
    for (int i=1; i<=10; i++) {
        String pubMsg = "{\"pubmsg\":"+ i + "}";
        int pubQoS = 0;
        MqttMessage message = new MqttMessage(pubMsg.g
message.setQos(pubQoS);
message.setRetained(false);

// Publish the message
System.out.println("Publishing to topic \\" + topic +
MqttDeliveryToken token = null;
try {
    // publish message to broker
    token = topic.publish(message);
    // Wait until the message has been delivered to the broker
    token.waitForCompletion();
    Thread.sleep(100);
} catch (Exception e) {
    e.printStackTrace();
}
}

// disconnect
try {
    // wait to ensure subscribed messages are delivered
    if (subscriber) {
        Thread.sleep(5000);
    }
    myClient.disconnect();
} catch (Exception e) {
    e.printStackTrace();
}
}
```

Bilan

Internet des objets

Objet ?

Microcontrôleur + capteurs

Interfaces de communication (I2C par exemple)

Mécanisme des GPIO

Lua : langage interprété proche de Javascript

RTOS : Real Time Operating System

Threads

DeepSleep

Wifi STA et AP

Serveur Web et interaction

MQTT et diffusion de messages

Accessible via Java / Javascript / C / C++ etc..