

# Laboratoires IoT avec PlatformIO

SmartComputerLab

## Contenu

<b>0. Introduction.....</b>	<b>2</b>
0.1 ESP32 Soc – une unité avancée pour les architectures IoT.....	3
0.2 Carte LOLIN D32.....	3
0.3 IoT DevKit une plate-forme de développement IoT.....	4
0.4 L'installation de PlatformIO.....	5
0.4.1 Installation de VSCode.....	5
0.4.2 Installer le package PlatformIO IDE pour VSCode.....	5
0.4.3 Premier démarrage de PlatformIO sur VSCode.....	6
0.4.4 Le menu PIO.....	7
0.4.5 Créer un nouveau projet (ESP32, ESP8266, ...)......	8
0.4.6 Décryptage du fichier platformio.ini.....	10
0.4.7 Edition du code.....	11
0.7.8 Premier exemple.....	12
<b>Laboratoire 1.....</b>	<b>15</b>
1.1 Premier exemple – l'affichage des données.....	15
A faire:.....	16
1.2 Deuxième exemple – capture et affichage des valeurs.....	17
1.2.1 Capture de la température/humidité par SHT21.....	17
A faire:.....	18
1.2.2 Capture de la luminosité par BH1750.....	19
1.2.3 Capture de la luminosité par MAX44009.....	20
1.2.5 Capture de la pression/température avec capteur BMP180.....	21
1.2.6 Capture de présence avec un capteur PIR SR602.....	22
<b>Laboratoire 2.....</b>	<b>23</b>
Communication en WiFi et broker MQTT.....	23
2.1 Client MQTT – envoi et réception des messages.....	23
A faire :.....	25
2.2 Simple broker MQTT avec ESP8266.....	26
A faire :.....	28
<b>Laboratoire 3 – WiFi avec WiFiManager et serveur ThingSpeak.com (.fr).....</b>	<b>29</b>
3.1 Introduction.....	29
3.1.1 Un programme de test – scrutation du réseau WiFi.....	29
3.2 Mode WiFi – STA, client WEB et serveur ThingSpeak.....	31
A faire.....	32
<b>Laboratoire 4 – communication longue distance avec LoRa (Long Range).....</b>	<b>39</b>
4.1 Introduction.....	39
4.1.1 Modulation LoRa.....	39
4.1.2 Paquets LoRa.....	40
4.2 Premier exemple – émetteur et récepteur des paquets LoRa.....	41
A faire :.....	43
4.3 onReceive() – récepteur des paquets LoRa avec une interruption.....	44
A faire :.....	45
<b>Laboratoire 5 - Développement de simples passerelles IoT.....</b>	<b>46</b>
5.1 Passerelle LoRa-ThingSpeak.....	46
5.1.1 Le principe de fonctionnement.....	46
5.1.2 Les éléments du code.....	46
5.1.3 Code complet pour une passerelle des paquets en format de base.....	49
A faire :.....	50
5.2 Passerelle LoRa – WiFi – broker MQTT.....	51
5.2.1 L'émetteur des messages MQTT sur LoRa.....	51
5.2.2 La passerelle des messages MQTT sur LoRa vers WiFi et broker MQTT.....	52
A faire :.....	53

## 0. Introduction

Dans les laboratoires IoT nous allons mettre en œuvre plusieurs architectures IoT intégrant les terminaux (T), les passerelles (*gateways* - G), et les serveurs-brokers (S,B) IoT type **ThingSpeak**, **MQTT**. Le développement sera réalisé sur les cartes **IoTDevKit** de **SmartComputerLab**.

Le kit de développement contient une carte de base "basecard" pour y accueillir une unité centrale et un ensemble de cartes d'extension pour les capteurs, les actionneurs et les modems supplémentaires. L'unité centrale est une carte équipée d'un **SoC ESP32** avec un modem **WiFi et Bluetooth**.

Dans l'introduction nous présentons l'environnement de développement **PlatformIO**.

**PlatformIO (PIO) IDE** intègre l'ensemble d'outils qui permet de programmer, de compiler, et de charger (flasher) les codes binaires sur nos cartes avec les SoC IoT - **ESP32/ESP8266**.

**Le premier laboratoire** permet de mettre en œuvre l'environnement de travail et de tester l'utilisation des capteurs connectés sur le bus I2C (température/humidité/luminosité/pression) et d'un afficheur. L'environnement de **PlatformIO** permet de chercher et d'installer les bibliothèques nécessaires pour l'interfaçage et l'exploitation des capteurs/afficheurs.

**Le deuxième laboratoire** est consacré à la prise en main du modem WiFi intégré dans la carte principale (ESP32-LOLIN). La communication WiFi en mode station (**STA**) permet d'envoyer les données – messages vers un broker **MQTT**. Un broker **MQTT** reçoit et renvoie les messages postés sur les **topic** données. Il ne possède pas d'une base de données pour y enregistrer les messages reçus.

L'utilisation d'une autre carte ESP basée sur le SoC ESP8266 permet le déploiement d'un micro-broker MQTT local fonctionnant en mode WiFi station (**STA**) et point d'accès (**AP**).

**Le troisième laboratoire** est dédié à l'utilisation du serveur **IoT ThingSpeak**. Les serveurs type **ThingSpeak** contiennent une base de données et offrent une interface graphique pour la visualisation des données enregistrées.

**ThingSpeak.com** est accessible gratuitement, mais sa fréquence de réception de messages et le nombre des messages sont limités.

**Le quatrième laboratoire** permet d'expérimenter avec les liens à longue distance (1Km). Il s'agit de la technologie (et modulation) **LoRa**. Le modem LoRa est intégré sur la carte d'extension de notre DevKit. Nous allons envoyer les données structurées d'un capteur géré par un terminal sur un lien LoRa vers une autre carte DevKit. La carte de destination va afficher les données reçues, ainsi que la force du signal associée au paquet reçu.

**Le cinquième laboratoire** permettra d'intégrer l'ensemble de liens WiFi et LoRa pour créer des applications complètes avec les terminaux et les passerelles **LoRa-WiFi** vers les brokers-serveurs **MQTT** et **ThingSpeak**.

Nous y développerons deux applications, une pour la passerelle type LoRa-WiFi (broker **MQTT**) et une pour la passerelle de type LoRa-WiFi (serveur **ThingSpeak**).

## 0.1 ESP32 Soc – une unité avancée pour les architectures IoT

ESP32 est une unité de microcontrôleur avancée conçue pour le développement d'architectures IoT. Un ESP32 intègre deux processeurs RISC 32-bit fonctionnant à 240 MHz et plusieurs unités de traitement et de communication supplémentaires, notamment un processeur ULP (*Ultra Low Power*), des modems WiFi / Bluetooth /BLE et un ensemble de contrôleurs E/S pour bus série (UART, I2C, SPI), ..). Ces blocs fonctionnels sont décrits ci-dessous dans la figure suivante.

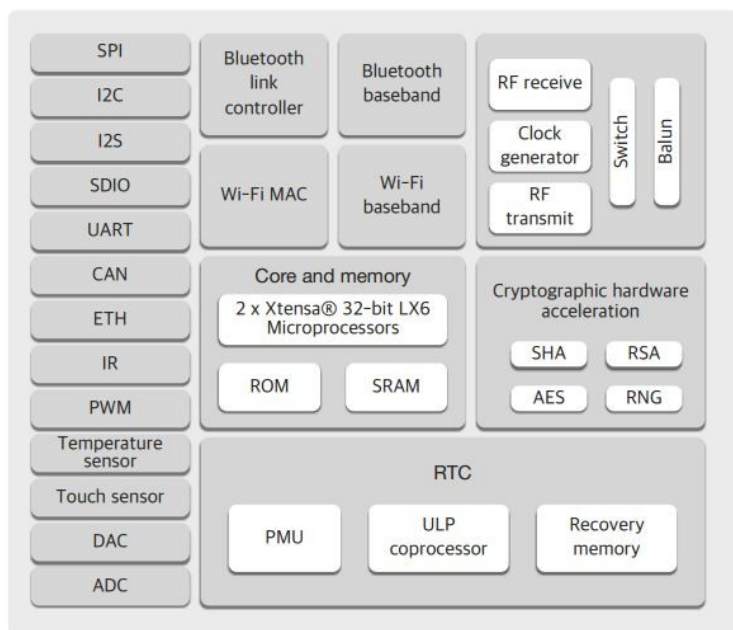


Figure 0.1 ESP32 SoC – architecture interne

## 0.2 Carte LOLIN D32

De nos jours, les SoC ESP32 sont intégrés dans un certain nombre de cartes de développement qui incluent des circuits supplémentaires et des modems de communication. Notre choix est la carte **ESP32-LOLIN D32** qui intègre une interface avec les batteries LiPo (3V7)

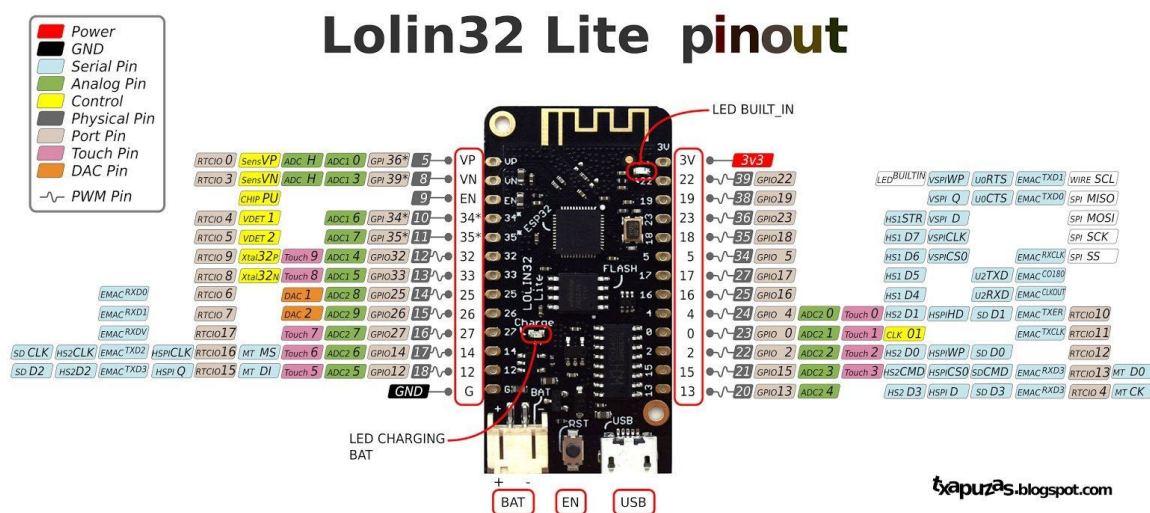


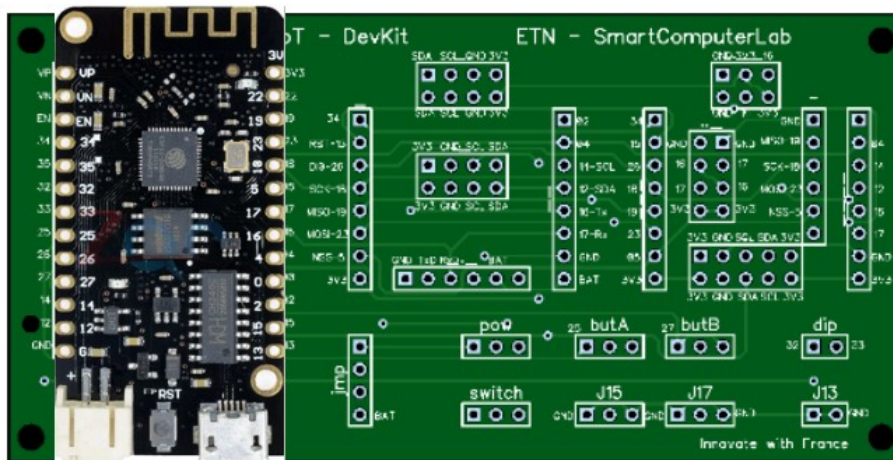
Figure 0.2 Carte MCU Lolin32 Lite (LOLIN D32) et son pinout

Comme nous pouvons le voir sur la figure ci-dessus, la carte Heltec expose 2x13 broches 2x18. Ces broches portent les bus **I2C** (14,12), **SPI** (18,19,23), **UART** (16,17) plus les signaux de contrôle ((NSS,RST,INT,..)

### 0.3 IoT DevKit une plate-forme de développement IoT

Une intégration efficace de la carte LOLIN sélectionnée dans les architectures IoT nécessite l'utilisation d'une plate-forme de développement telle que IoT DevKit proposée par **SmartComputerLab**.

L'**IoTDevKit** est composé d'une carte de base et d'un grand nombre de cartes d'extension conçues pour l'utilisation efficace des bus de connexion et de tous les types de capteurs et d'actionneurs.

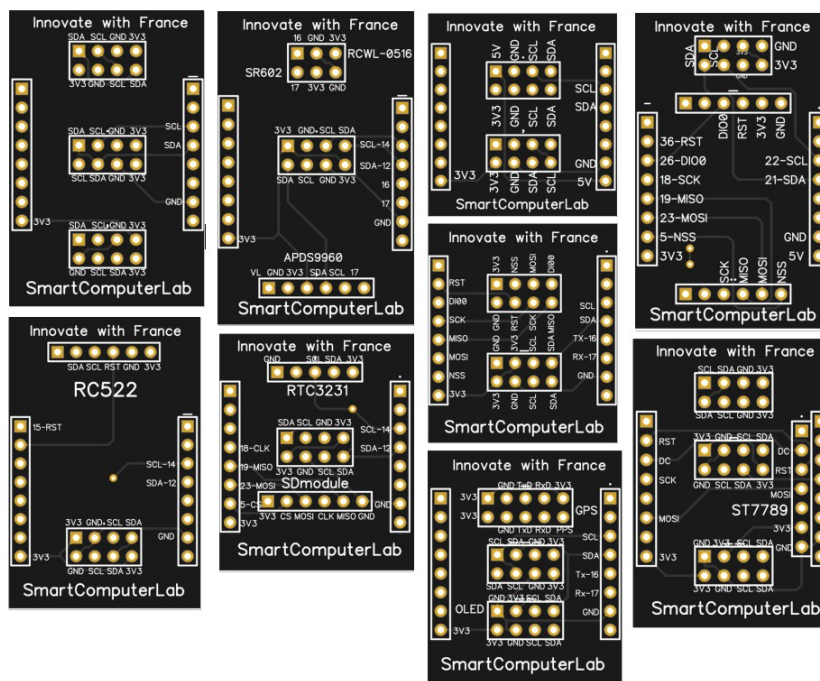


**Figure 0.3** Carte de base pour **Lolin32 Lite (LOLIN D32)** et ses interfaces intégrées

La carte de base peut accueillir directement plusieurs types de capteurs ou modems de communication. Pour y connecter un ensemble plus complet de capteurs/modems/afficheurs on utilise les cartes d'extension.

La carte de base intègre les emplacements pour les commutateurs (**dip**) et les boutons (**butA, butB**) poussoirs. Le chevalier (**jump**) permet de connecter un multimètre et d'effectuer les mesures du courant. En mode faible consommation le courant descend à quelques dizaines de micro-ampères.

Ci dessous quelques exemples de cartes d'extension.



**Figure 0.4** Cartes d'extension pour les différents composants IoT: capteurs, afficheurs, modems, ..

## 0.4 L'installation de PlatformIO

Pour nos développements et nos expérimentations nous utilisons l'IDE **PlatformIO** comme outils de programmation et de chargement des programmes dans la mémoire flash de l'unité centrale.

**PlatformIO** est un écosystème complet pour développer des objets connectés qui peut remplacer l'ancestral IDE Arduino. PIO est disponible sous la forme d'une extension disponible pour la plupart des éditeurs de code (Atom, VSCode...).

PIO est un environnement de développement professionnel complet qui propose de très nombreux outils :

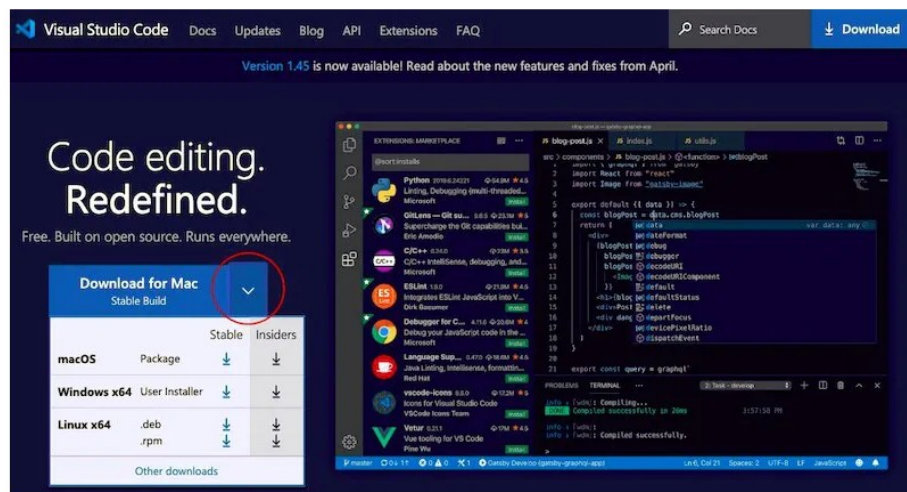
- un compilateur multi-plateforme,
- un gestionnaire de librairie,
- un debugger, (nécessite une carte ESP-prog avec JTAG)
- un système de test unitaire,

### 0.4.1 Installation de VSCode

Rendez-vous sur [la page officielle](#) de VSCode pour télécharger et installer la version qui correspond à votre environnement.

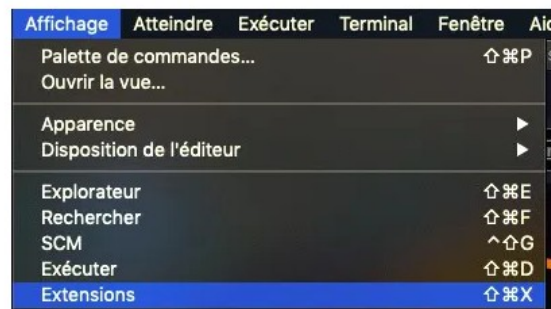
Normalement, le navigateur reconnaît automatiquement votre système d'exploitation. Si ce n'est pas le cas, cliquez sur la flèche située à droite du bouton Download pour choisir la version à télécharger

- Windows : installer ou ZIP
- macOS : package dmg
- Linux 32-bits : .deb, .rpm, .tag.gz
- Linux 64-bits : .deb, .rpm, .tag.gz



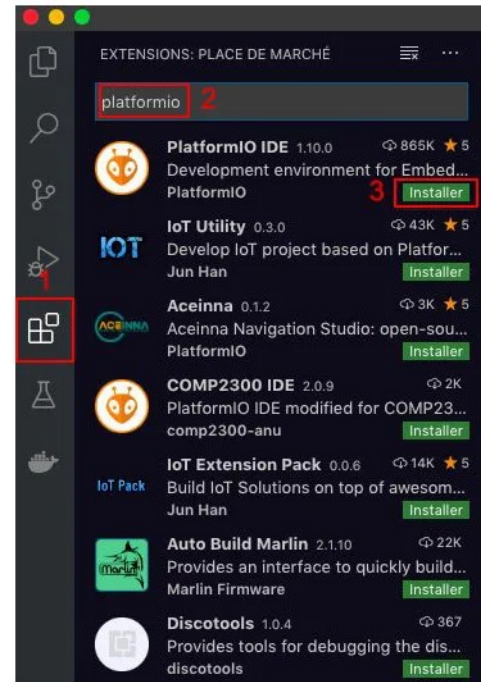
### 0.4.2 Installer le package PlatformIO IDE pour VSCode

**VSCode** dispose d'un gestionnaire d'extension (plugin) que l'on ouvre via le menu **Affichage -> Extension** ou directement depuis l'icône située dans la barre latérale.





Saisissez **platformio** dans le champ de recherche. Cliquez sur **Install** pour démarrer l'installation du plugin et des dépendances.

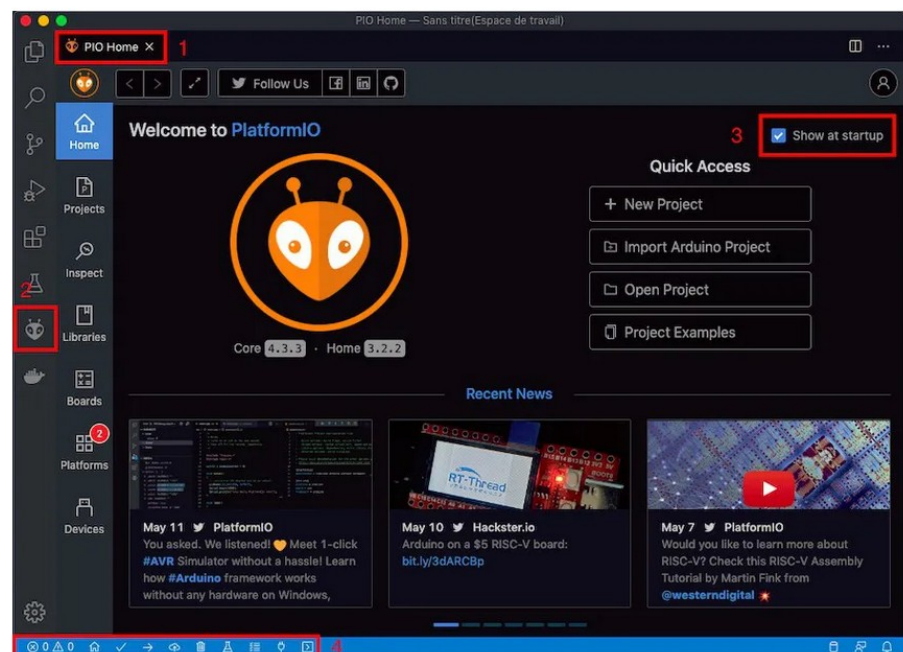


Le processus d'installation se déroule en arrière plan, c'est un peu perturbant lorsqu'on découvre VSCode. Une fenêtre située en bas à droite de l'écran permet de suivre le bon déroulement de l'installation. C'est assez rare mais si vous rencontrez un problème d'installation (ou un blocage), il suffit de relancer VSCode pour reprendre le processus d'installation.

### 0.4.3 Premier démarrage de PlatformIO sur VSCode

A la fin de l'installation, il n'est pas nécessaire de redémarrer l'éditeur. La page d'accueil de PIO s'ouvre dans un nouvel onglet ①. Cette page ralentit fortement le démarrage de VSCode et n'apporte rien d'utile. Je vous conseille de désactiver son ouverture au démarrage en décochant l'option **Show at startup** ③.

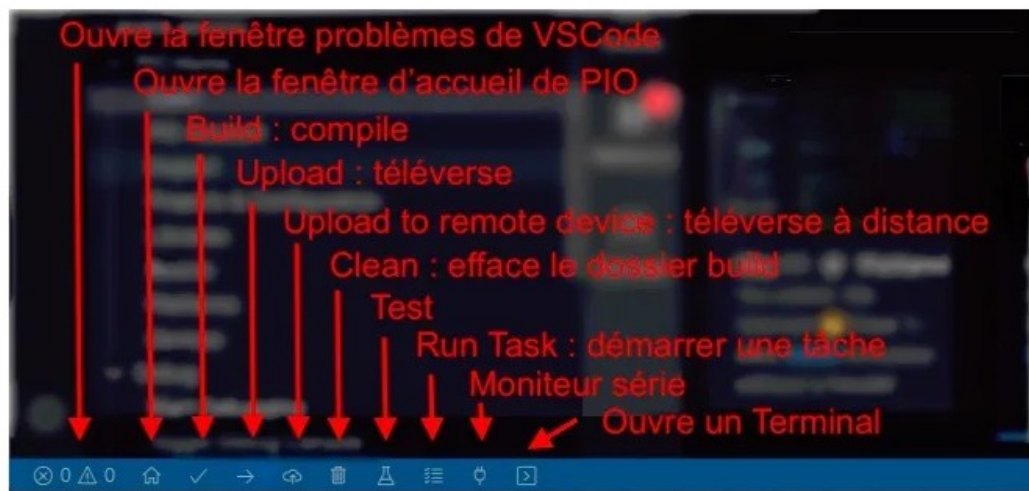
Une nouvelle icône sous la forme d'une tête de fourmi fait son apparition dans la barre latérale ②. Elle permet d'accéder à toutes les fonctionnalités de PIO. Nous allons les détailler un peu plus tard.



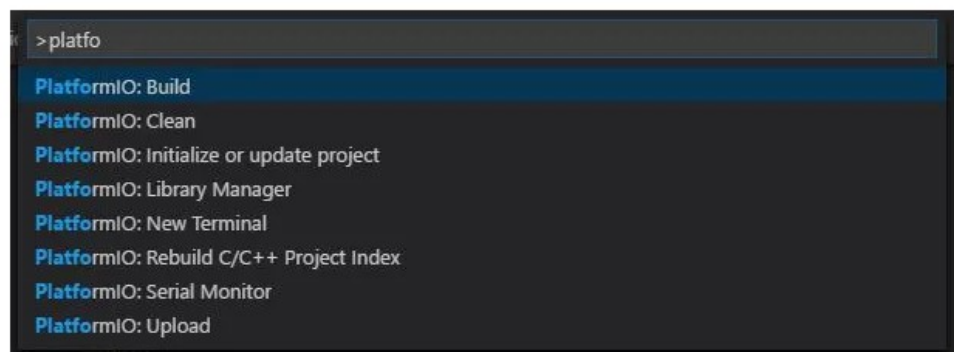
Enfin, une **mini barre** d'outil apparaît dans le bas de l'écran. C'est une version allégée du menu PIO .

Elle regroupe les fonctions suivantes

- **Errors** signale les problèmes de compilation
- **Home** ouvre la fenêtre d'accueil de PIO. Utile lorsqu'on veut importer ou créer un nouveau projet
- **Build** compile le code du projet. Permet de vérifier s'il y a des erreurs
- **Upload** compile et téléverse le projet. La détection est automatique mais il est aussi possible de spécifier le port dans le fichier de configuration
- **Upload to remote device** idem mais sur un MCU distant. Un compte PIO (payant ou limité dans l'offre gratuite) est nécessaire
- **Clean** supprime le dossier build. Ne pas hésiter en cas de problème. N'a aucun impact sur le code source du projet
- **Test** pour tester le code avant de le compiler
- **Run Task** ouvre la palette de commande de VSCode
- **Serial Monitor** ouvre le moniteur série
- **Terminal** ouvre un Terminal directement dans VSCode ou Power Shell sous Windows. On est directement positionné à la racine du projet



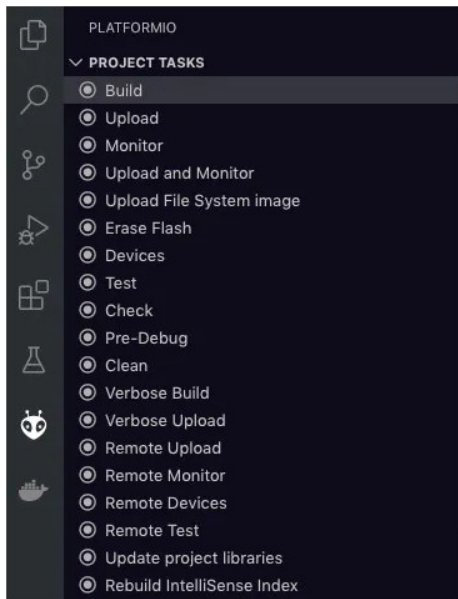
Pour ceux qui préfèrent utiliser les raccourcis clavier, vous pouvez convoquer la palette avec la combinaison de touche **Ctrl + Shift + P** (ou **Cmd + ↑ + P** sur macOS). Saisissez ensuite le mot clé **platformio** pour afficher toutes les commandes disponibles



#### 0.4.4 Le menu PIO

Revenons au menu PIO qui est le moyen le plus simple et le plus complet pour utiliser **PIO**. Il est en permanence accessible depuis la barre latérale. Elle regroupe toutes les fonctions de **PIO**.

Voici les commandes les plus utiles :



**Build** compiler

**Upload** compile et téléverse le programme

**Monitor** ouvre le moniteur série

**Upload and monitor** compile, téléverse et ouvre le moniteur série

**Upload File System image** téléverse les fichiers du dossier data stockés au format SPIFFS ou LittleFS.

**Erase Flash** vide la mémoire flash de la carte de développement

**Devices** Liste les devices connectés

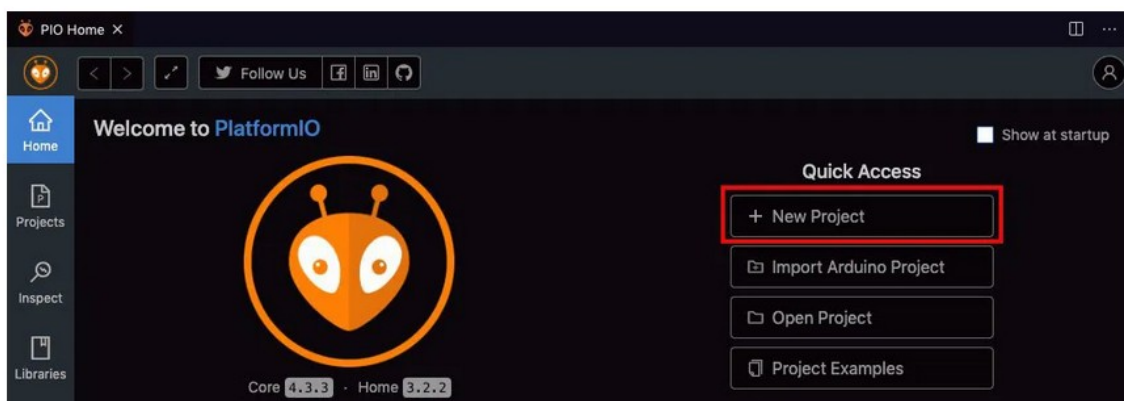
**Clean** efface le dossier build

**Update project libraries** met à jour les librairies utilisées par le projet

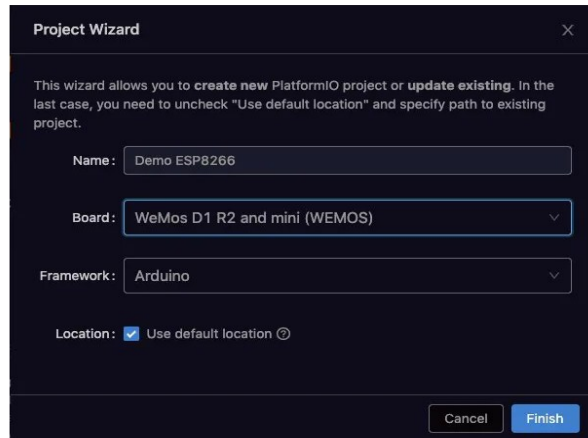
#### 0.4.5 Créer un nouveau projet (ESP32, ESP8266, ...)

Il est temps de passer à un petit exemple pour tester tout ça. Ouvrez la fenêtre d'accueil de **PlatformIO - PIO** (si nécessaire) et cliquez sur **+ New Project** pour ouvrir l'assistant de création de projet

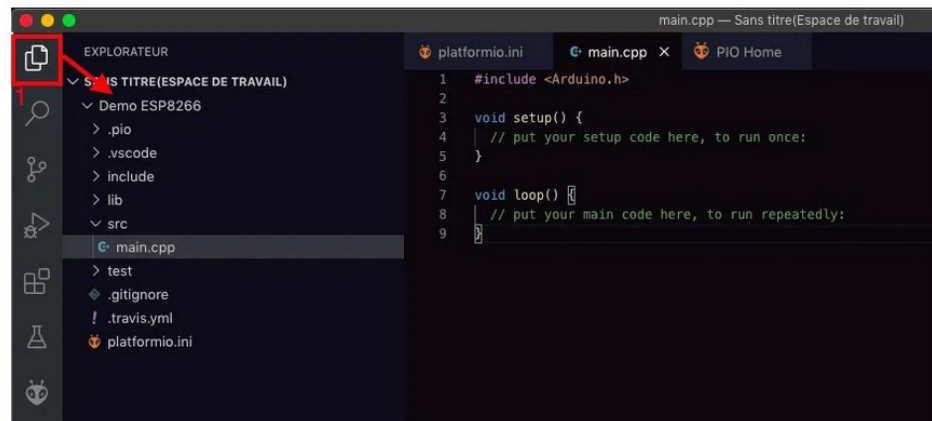
Nommer le projet puis sélectionnez la carte de développement dans la liste. La liste est impressionnante. Vous pouvez saisir les premières lettres du fabricant (LOLIN., Heltec., WeMos, TTGO..), celle de la carte (D1 mini), le type (**ESP32**, **ESP8266**...) pour filtrer les cartes.





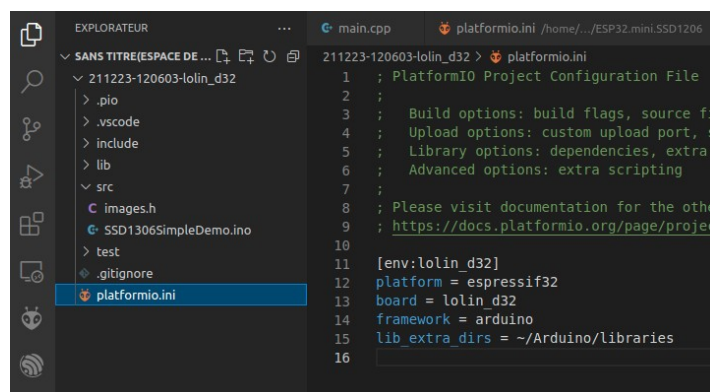


Laissez **Arduino** dans le framework. En fonction de votre carte, vous pourrez opter pour un autre framework, mais dans ce cas il faudra vous reporter à la documentation de ce dernier pour la programmation. Enfin, choisissez le répertoire de création du projet. Si vous cochez **Use Default Location**, le projet sera créé dans le dossier **Documents/PlatformIO/Projects**. Le nom de répertoire portera le nom du projet. Cliquez sur Finish pour lancer l'initialisation du projet. L'opération ne dure qu'une poignée de secondes. Le projet est maintenant accessible depuis l'explorateur.



Le dossier contient plusieurs dossiers et fichiers de configuration.

- **.pio** est un dossier (caché) qui contient les builds. Ce sont les fichiers binaires générés par le compilateur. Un sous-dossier est créé par cible (carte de développement)
- **lib** est le dossier dans lequel seront installées les bibliothèques nécessaires au projet. Cela permet de mieux gérer les problèmes de versions d'un projet à l'autre. Par contre, attention à la consommation d'espace disque.
- **src**. Ce dossier contient le code source de votre projet. C'est votre dossier de travail
- **platformio.ini** est le fichier de configuration de PIO



## 0.4.6 Décryptage du fichier `platformio.ini`

Découvrons maintenant pour décrypter ce que contient le fichier de configuration `platformio.ini` qui se trouve à la racine du projet.

La force de PIO est de pouvoir compiler un même code (projet) vers autant de cibles (cartes de développement, MCU) que l'on souhaite. La configuration de chaque carte se fait par bloc qui commence par la clé `env:` entre crochets, par exemple `[env:esp12e]` pour la LoLin d1 mini. Il faut 3 paramètres pour définir complètement une carte :

- **platform** qui correspond au SoC utilisé par la carte (ESP32, ESP8266, Atmel AVR, STM32...). La liste complète est [ici](#)
- **board**, la carte de développement. La liste complète se trouve [ici](#)
- **framework**, l'environnement logiciel qui fera fonctionner le code du projet. Attention, chaque SoC n'est compatible qu'avec un nombre limité de framework. La liste se trouve [ici](#).
- **lib\_extra\_dirs** = `~/Arduino/libraries`, chemin d'accès aux libraries

On pourra ensuite spécifier d'autres paramètres, par exemple **upload\_port**, spécifier le port **COM**

- `/dev/ttyUSB0` – port série sur les systèmes basés sur Linux (ou macOS)
- `COM3` – port série sur Windows
- `192.168.0.13` – adresse IP pour la mise à jour sans fil en WiFi ([OTA](#))
- **upload\_speed** – spécifier la vitesse de transfert en baud
- **monitor\_speed** – vitesse du moniteur série

Pour ajouter une nouvelle carte de développement, je vous conseille de récupérer directement la configuration de votre carte sur le [Wiki officiel](#) plutôt que d'utiliser l'assistant de configuration. Plus de 1000 cartes de développement sont déjà répertoriées. Vous n'aurez pas à tâtonner pour trouver les bons réglages comme pour cette carte [Heltec ESP32](#) avec connectivité LoRa qui existe en 2 versions.

## Configuration

Please use `heltec_wifi_lora_32_V2` ID for `board` option in "`platformio.ini`" (Project Configuration File):

```
[env:heltec_wifi_lora_32_V2]
platform = espressif32
board = heltec_wifi_lora_32_V2
```

You can override default Heltec WiFi LoRa 32 (V2) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `heltec_wifi_lora_32_V2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

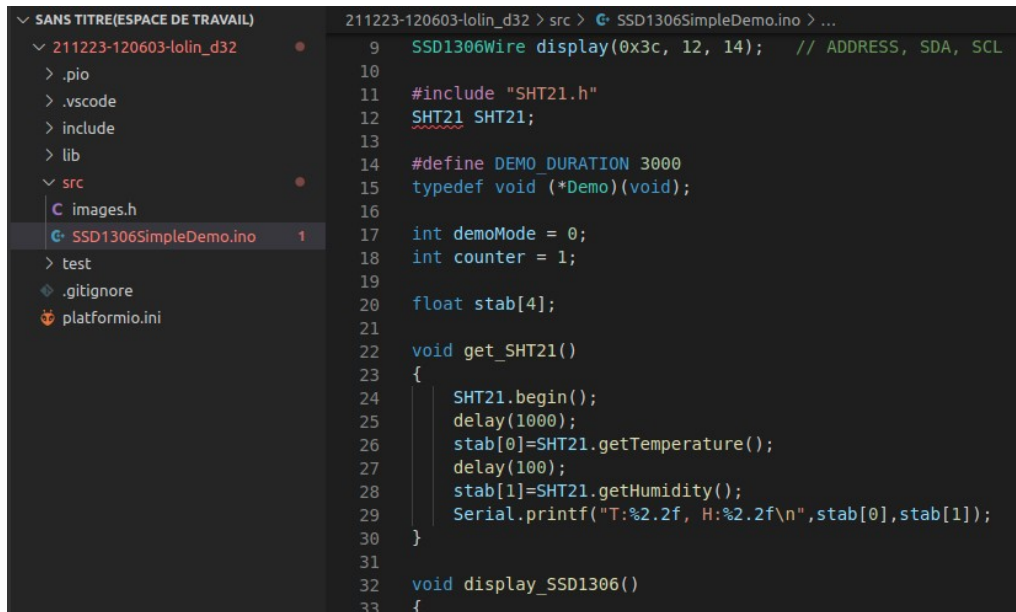
```
[env:heltec_wifi_lora_32_V2]
platform = espressif32
board = heltec_wifi_lora_32_V2

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

## 0.4.7 Edition du code

Voici un exemple du code source (projet **Arduino** avec extension **.ino**) :



```
211223-120603-lolin_d32 > src > SSD1306SimpleDemo.ino > ...
9  SSD1306Wire display(0x3c, 12, 14); // ADDRESS, SDA, SCL
10
11  #include "SHT21.h"
12  SHT21 SHT21;
13
14  #define DEMO_DURATION 3000
15  typedef void (*Demo)(void);
16
17  int demoMode = 0;
18  int counter = 1;
19
20  float stab[4];
21
22  void get_SHT21()
23  {
24      SHT21.begin();
25      delay(1000);
26      stab[0]=SHT21.getTemperature();
27      delay(100);
28      stab[1]=SHT21.getHumidity();
29      Serial.printf("T:%2.2f, H:%2.2f\n",stab[0],stab[1]);
30  }
31
32  void display_SSD1306()
33  {
```

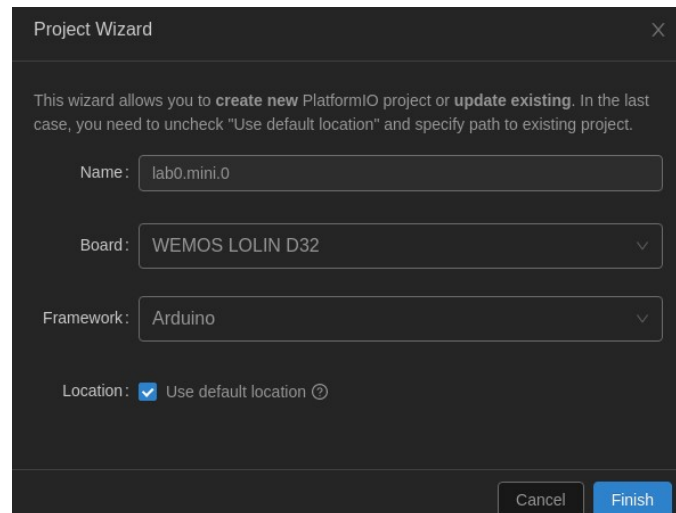
### Compilation et chargement (flash) du code :

```
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash will be erased from 0x00001000 to 0x00005fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Flash will be erased from 0x0000e000 to 0x0000ffff...
Flash will be erased from 0x00010000 to 0x00048fff...
Compressed 17104 bytes to 11191...
Writing at 0x00001000... (100 %)
Wrote 17104 bytes (11191 compressed) at 0x00001000 in 0.5 seconds (effective 290.5 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 128...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.1 seconds (effective 432.3 kbit/s)...
Hash of data verified.
Compressed 8192 bytes to 47...
Writing at 0x0000e000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.1 seconds (effective 510.2 kbit/s)...
Hash of data verified.
Compressed 231952 bytes to 121556...
Writing at 0x00010000... (12 %)
Writing at 0x0001cba7... (25 %)
Writing at 0x00024858... (37 %)
Writing at 0x0002ba67... (50 %)
Writing at 0x00033fde... (62 %)
Writing at 0x00039887... (75 %)
Writing at 0x0004044f... (87 %)
Writing at 0x00045f61... (100 %)
Wrote 231952 bytes (121556 compressed) at 0x00010000 in 3.0 seconds (effective 623.8 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
===== [SUCCESS] Took 7.41 seconds
=====

Le terminal sera réutilisé par les tâches, appuyez sur une touche pour le fermer.
```

## 0.7.8 Premier exemple

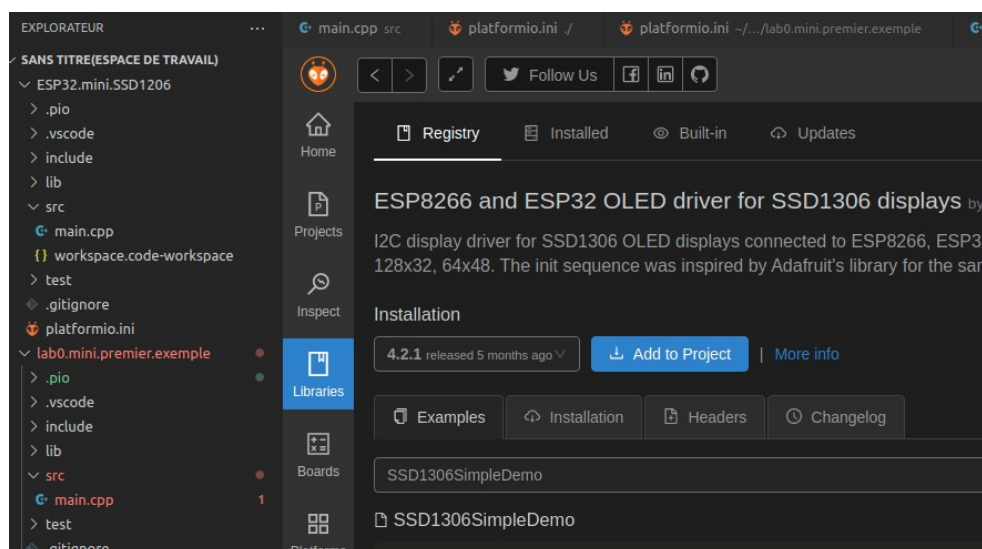


**PlatformIO Project Configuration File : (platformio.ini)**

```
; Build options: build flags, source filter
; Upload options: custom upload port, speed and extra flags
; Library options: dependencies, extra library storages
; Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html
```

```
[env:lolin_d32]
platform = espressif32
board = lolin_d32
framework = arduino
lib_deps =
    https://github.com/markbeeee/SHT21.git
upload_port = /dev/ttyUSB0
```

La bibliothèque **SHT21** est ajoutée directement dans le fichier d'initialisation.  
Un autre moyen d'ajouter une nouvelle bibliothèque passe par l'interface graphique dans le **Home** de **PlatformIO**. La figure ci-dessous montre l'ajout de la bibliothèque **SSD1306Wire** par le biais de l'interface graphique.





## Le code source : main.cpp

```
#include <Arduino.h>
#include <Wire.h>
#include "SSD1306Wire.h" // legacy:
#include "SSD1306.h"
SSD1306Wire display(0x3c, 12, 14);
#include "SHT21.h" // data path to github declared in platformio.ini
SHT21 SHT21;

#define DEMO_DURATION 3000
typedef void (*Demo)(void);

int demoMode = 0;
int counter = 1;
float stab[4];

void get_SHT21()
{
    SHT21.begin();
    delay(1000);
    stab[0]=SHT21.getTemperature();
    delay(100);
    stab[1]=SHT21.getHumidity();
    Serial.printf("T:%2.2f, H:%2.2f\n",stab[0],stab[1]);
}

void display_SSD1306()
{
    char buff[32];
    display.init();
    //display.flipScreenVertically();
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.setFont(ArialMT_Plain_16);
    display.drawString(0, 0, "ETN - IoT DevKit");
    display.setFont(ArialMT_Plain_16);
    display.drawString(0, 20, "Temp and Humi");
    sprintf(buff,"T:%2.2f,H:%2.2f",stab[0],stab[1]);
    display.drawString(0, 46, buff);
    display.display();
}

void setup() {
    Serial.begin(9600);
    Wire.begin(12,14);
    Serial.println();
    // Initialising the UI will init the display too.
}

void loop()
{
    Serial.println("in the loop");
    get_SHT21(); delay(2000);
    display_SSD1306();
    delay(2000);
}
```

```
55 {
56   Serial.println("in the loop");
57   oet SHT21(): delay(2000);

```

PROBLÈMES 3 SORTIE CONSOLE DE DÉBOGAGE TERMINAL

```
Linking .pio/build/lolin_d32/firmware.elf
Retrieving maximum program size .pio/build/lolin_d32/firmware.elf
Checking size .pio/build/lolin_d32/firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM: [      ] 4.4% (used 14312 bytes from 327680 bytes)
Flash: [==   ] 17.7% (used 231838 bytes from 1310720 bytes)
Building .pio/build/lolin_d32/firmware.bin
esptool.py v3.1
Merged 1 ELF section
===== [SUCCESS] Took 8.97 seconds =====

Le terminal sera réutilisé par les tâches, appuyez sur une touche pour le fermer.
```

Le résultat du **build** affiché dans le terminal PIO.

Maintenant il s'agit de télécharger (flasher) le code binaire sur la carte.

Voici l'affichage dans le terminal connecté à la carte :

```
printable, send_on_enter, time
--- More details at https://bit.ly/pio-monitor-filters
--- Miniterm on /dev/ttyUSB0 9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
T:22.32, H:40.35
in the loop
T:22.33, H:40.22
in the loop
T:22.34, H:40.26
in the loop
T:22.34, H:40.20
in the loop
T:22.33, H:40.16
```

# Laboratoire 1

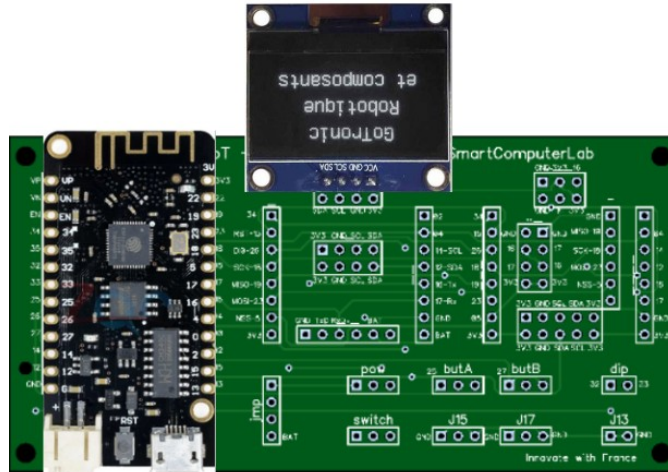
## 1.1 Premier exemple – l'affichage des données

Dans cet exercice nous allons simplement afficher un titre et 2 valeurs numériques sur l'écran OLED ajouté à notre ESP32.

Vous pouvez créer un nouveau projet **PlatformIO** type **Arduino**, par exemple, **Lab1.1.oled**

### Remarque :

Ce projet peut exploiter le fichier de configuration élaboré dans **Lab0**.



**Figure 1.1** Ecran **OLED** ajouté à la carte ESP32 (**LOLIN D32**)

Vous devez installer la bibliothèque **OLED** compatible avec ESP32 (recherchez – **SSD1306.h** dans le **libraries**)

### Le code

```
#include <Wire.h>
#include "SSD1306Wire.h"
SSD1306Wire display(0x3c, 12, 14);

void display_SSD1306(float d1,float d2,float d3,float d4)
{
    char buff[64];
    display.init();
    //display.flipScreenVertically();
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.setFont(ArialMT_Plain_16);
    display.drawString(0, 0, "ETN - IoT DevKit");
    display.setFont(ArialMT_Plain_10);
    sprintf(buff,"d1: %.2f, d2: %.2f\nd3: %.2f, d4: %.2f",d1,d2,d3,d4);
    display.drawString(0, 22, buff);
    display.drawString(20, 52, "SmartComputerLab");
    display.display();
}

void setup() {
    Serial.begin(9600);
    Wire.begin(12,14);    // SDA - 12,  SCL - 14
    Serial.println("Wire started");
}
```

```
float v1=0.0,v2=0.0,v3=0.0,v4=0.0;

void loop()
{
  Serial.println("in the loop");
  display_SSD1306(v1,v2,v3,v4);
  v1=v1+0.1;v2=v2+0.2;v3=v3+0.3;v4=v4+0.4;
  delay(2000);
}
```

**A faire:**

**Compléter, compiler, et charger ce programme.**



## 1.2 Deuxième exemple – capture et affichage des valeurs

### 1.2.1 Capture de la température/humidité par SHT21

Dans cet exercice nous allons lire les valeurs fournies par le capteur Température/Humidité **SHT21** et afficher les 2 valeurs sur l'écran OLED ajouté sur la carte ESP32. Le capteur **SHT21** doit également être connecté sur le bus **I2C**.

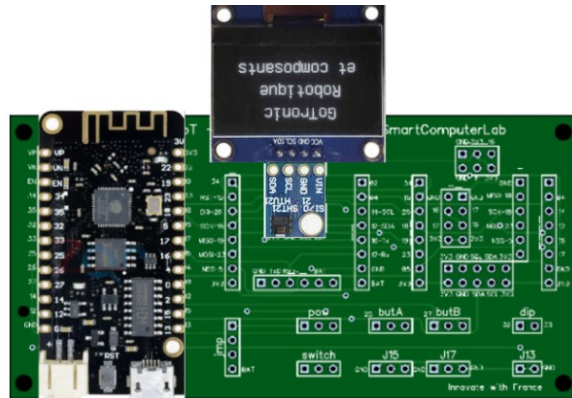


Figure 1.5. IoT DevKit avec un capteur de température/humidité SHT21 et l'écran OLED.

#### Code :

Avant de commencer le codage il faut télécharger et installer la bibliothèque **SHT21.h** compatible avec notre carte. Elle se trouve dans le **github**: [e-radionicacom/SHT21-Arduino-Library](https://github.com/e-radionicacom/SHT21-Arduino-Library)

```
#include <Wire.h>
#include "SSD1306Wire.h"
SSD1306Wire display(0x3c, 12, 14);
#include "SHT21.h"
SHT21 SHT21;
float stab[4]= {0.0,0.0,0.0,0.0};

void get_SHT21()
{
    SHT21.begin();
    delay(1000);
    stab[0]=SHT21.getTemperature();
    delay(100);
    stab[1]=SHT21.getHumidity();
    Serial.printf("T:%2.2f, H:%2.2f\n",stab[0],stab[1]);
}

void display_SSD1306(float d1,float d2,float d3,float d4)
{
    char buff[64];
    display.init(); //display.flipScreenVertically();
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.setFont(ArialMT_Plain_16);
    display.drawString(0, 0, "ETN - IoT DevKit");
    display.setFont(ArialMT_Plain_10);
    sprintf(buff,"T: %2.2f, H: %2.2f\n d3: %2.2f, d4: %2.2f",d1,d2,d3,d4);
    display.drawString(0, 22, buff);
    display.drawString(20, 52, "SmartComputerLab");
    display.display();
}
```

```

void setup() {
  Serial.begin(9600);
  Wire.begin(12,14);
  Serial.println();
}

void loop()
{
  Serial.println("in the loop");
  // à compléter
  delay(2000);
}

```

### A faire:

1. Créer un projet **PlatformIO**, compléter, compiler, et charger ce programme.
2. Tester le programme suivant (**I2CScan**) pour vérifier que les deux dispositifs (l'écran **SSD1306** et le capteur **SHT21**) sont visibles avec leurs adresses correspondantes (**0x3C** et **0x40**).

```

#include <Wire.h>

void I2Cscan()
{
  byte address; int nDevices; delay(200);
  Serial.println("Scanning...");
  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    Wire.beginTransmission(address);
    if(! Wire.endTransmission()) // active address found
    {
      Serial.print("I2C device found at address 0x");
      if (address<16) Serial.print("0");
      Serial.print(address,HEX); Serial.println("  !");
      nDevices++;
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");
}

void setup()
{
  Serial.begin(9600);
  Wire.begin(12,14,400000); // SDA, SCL on ESP32, 400 kHz rate
  delay(1000); Serial.println();Serial.println();
  I2Cscan(); delay(1000);
}

void loop(){ }

```

## 1.2.2 Capture de la luminosité par BH1750

Dans cet exercice nous utilisons le capteur de la luminosité **BH1750**

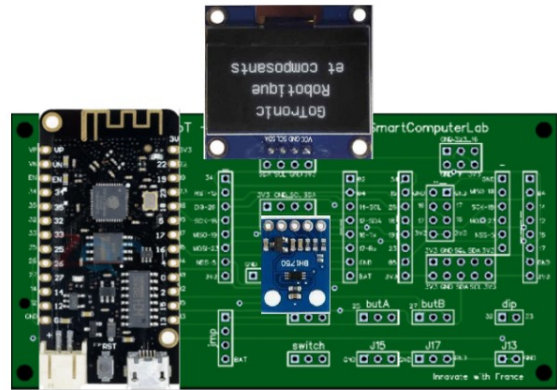


Figure 1.3. IoT DevKit avec le capteur Luminosité **BH1750**

**Attention** : Avant la compilation il faut installer la bibliothèque **BH1750.h**

```
#include <Wire.h>
#include <BH1750.h>
BH1750 lightMeter;

void setup() {
  Wire.begin(12, 14);
  Serial.begin(9600);
  lightMeter.begin();
  Serial.println("Running...");
  delay(1000);
}

void loop() {
  uint16_t lux = lightMeter.readLightLevel();
  delay(1000);
  Serial.print("Light: ");
  Serial.print(lux);
  Serial.println(" lx");
  delay(1000);
}
```

**A faire :**

1. Tester le programme
2. Ajouter l'affichage sur l'écran OLED.

### 1.2.3 Capture de la luminosité par MAX44009

Dans cet exemple nous utilisons un capteur de luminosité type **MAX44009** (GY-49). Ce capteur est connectée comme le capteur BH1750 sur le bus I2C.

Nous communiquons avec ce capteurs **directement** par les trames **I2C** ce qui permet de mieux comprendre le fonctionnement de ce bus.



Figure 1.4. Capteur de luminosité **MAX44009**

#### Code Arduino :

```
#include <Wire.h>
#define Addr 0x4A

void setup()
{
  Wire.begin(12,14);
  Serial.begin(9600);
  Wire.beginTransmission(Addr);
  Wire.write(0x02);Wire.write(0x40);
  Wire.endTransmission();
  delay(300);
}

void loop()
{
  unsigned int data[2];
  Wire.beginTransmission(Addr);
  Wire.write(0x03);
  Wire.endTransmission();
  // Request 2 bytes of data
  Wire.requestFrom(Addr, 2);
  // Read 2 bytes of data luminance msb, luminance lsb
  if (Wire.available() == 2)
  {
    data[0] = Wire.read();data[1] = Wire.read();
  }
  // Convert the data to lux
  int exponent = (data[0] & 0xF0)>>4;
  int mantissa = ((data[0] & 0x0F) << 4) | (data[1]& 0x0F);
  float luminance = pow(2, exponent) * mantissa * 0.045;
  Serial.print("Ambient Light luminance :");
  Serial.print(luminance);
  Serial.println(" lux");
  delay(500);
}
```

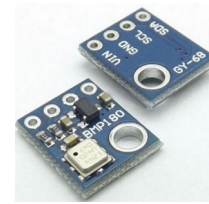
#### A faire :

3. Tester le programme
4. Ajouter l'affichage sur l'écran OLED.



## 1.2.5 Capture de la pression/température avec capteur BMP180

Le capteur BMP180 permet de capter la pression atmosphérique et la température. Sa précision est seulement de +/- 100 Pa et +/-1.0C.



**Figure 1.6.** Capteur de pression/température BMP180

La valeur standard de la pression atmosphérique est :

$$101\,325\text{ Pa} = 1,013\,25\text{ bar} = 1\text{ atm}$$

**Code platformio.ini :**

```
[env:lolin_d32]
platform = espressif32
board = lolin_d32
framework = arduino
lib_deps =
    https://github.com/adafruit/Adafruit-BMP085-Library.git
```

**Code main.cpp :**

```
#include <Arduino.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_BMP085.h>
Adafruit_BMP085 bmp;

void setup() {
    Serial.begin(9600);
    Wire.begin(12,14);
    bmp.begin();
}

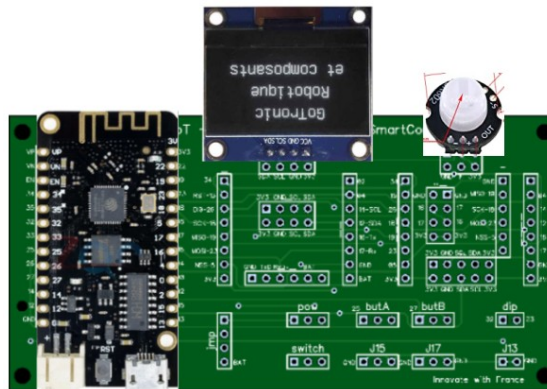
void loop() {
    Serial.print("Temperature = ");
    Serial.print(bmp.readTemperature()); Serial.println(" *C");
    Serial.print("Pressure = "); Serial.print(bmp.readPressure());
    Serial.println(" Pa");
    // Calculate altitude assuming 'standard' barometric
    // pressure of 1013.25 millibar = 101325 Pascal
    Serial.print("Altitude = ");
    Serial.print(bmp.readAltitude());
    Serial.println(" meters");
    // you can get a more precise measurement of altitude
    // if you know the current sea level pressure which will
    // vary with weather and such. If it is 1015 millibars
    // that is equal to 101500 Pascals.
    Serial.print("Real altitude = ");
    Serial.print(bmp.readAltitude(101500)); Serial.println(" meters");
    Serial.println();
    delay(500);
}
```

**A faire :**

1. Complétez le programme ci-dessus afin d'afficher les données de la température et pression atmosphérique sur l'écran OLED

## 1.2.6 Capture de présence avec un capteur PIR SR602

Dans l'exemple suivant nous utilisons un capteur de type PIR - **SR602** qui permet de détecter la présence d'une personne en mouvement. Il s'agit d'un capteur simple qui signale l'événement par le basculement de son signal de sortie.



**Figure 1.7.** Capteur de mouvement – SR602

Dans le code ci-dessous le changement de la valeur du signal de sortie provoque une interruption captée sur la broche associée à la fonction `pinChanged()` qui s'exécute de la façon asynchrone par rapport au déroulement du programme.

```
#define PIR 0 // SIG
bool MOTION_DETECTED = false;

void pinChanged()
{
    MOTION_DETECTED = true;
}

void setup()
{
    Serial.begin(9600);
    pinMode(PIR, INPUT_PULLDOWN);
    attachInterrupt(PIR, pinChanged, RISING);
}

int counter=0;

void loop()
{
    int i=0;
    if(MOTION_DETECTED)
    {
        Serial.println("Motion detected.");
        delay(1000);counter++;
        MOTION_DETECTED = false;
        Serial.println(counter);
    }
}
```

**Remarque :** La carte doit être réactivée par le bouton **RST**

### A faire :

1. Complétez le programme ci-dessus afin d'afficher la valeur du compteur de détection des mouvements sur l'écran OLED

## Laboratoire 2

### Communication en WiFi et broker MQTT

MQTT est l'un des protocoles IoT les plus importants, largement utilisé dans les projets IoT pour connecter des cartes telles que ESP8266, ESP32 au courtier (**broker**) cloud MQTT. Cette introduction suppose que vous connaissez MQTT et que vous connaissez les aspects de base tels que le courtier MQTT, les rubriques (**topics**) MQTT et l'architecture de publication (**publish**) et d'abonnement (**subscribe**).

Si vous débutez dans ce protocole IoT, il est utile d'en savoir plus sur **MQTT** en lisant une description technique.

Ce laboratoire explique comment développer un client ESP32 MQTT pour publier des messages MQTT et s'abonner aux rubriques MQTT. Nous allons utiliser la bibliothèque **MQTT.h** pour connecter l'ESP32 au courtier (**broker**) **MQTT**. Pour mieux comprendre comment utiliser un client ESP32 MQTT, nous utiliserons un exemple de publication ESP32 MQTT se connectant au courtier **broker.emqx.io**.

Le client enverra les données dans les rubriques (topics) MQTT. De plus, nous allons découvrir comment s'abonner aux rubriques MQTT pour que l'ESP32 puisse recevoir des données.

#### 2.1 Client MQTT – envoi et réception des messages

Voici un exemple complet du code qui permet de se connecter au broker, d'envoyer – publier et recevoir - s'abonner à un **topic** donnée. Le code fonctionne sur la carte ESP32.

```
#include <Arduino.h>

#include <WiFi.h>
#include <MQTT.h>
#include <Wire.h>
#include "SSD1306Wire.h"
#include "SSD1306.h"
SSD1306Wire display(0x3c, 12, 14); // SDA, SCL

const char* ssid = "PhoneAP";
const char* pass = "smartcomputerlab";

// const char* ssid = "MyMQTT-5";
// const char* pass = "smartcomputerlab";

const char* mqttServer = "broker.emqx.io";
//const char* mqttServer = "192.168.5.1";

WiFiClient net;
MQTTClient client;

typedef struct
{
  char topic[32];
  char mess[32];
} MQTTPack; // MQTT packet

void display_MQTT(char *top, char *mes)
{
  char buff[32];
  display.init();
  display.flipScreenVertically();
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(8, 0, "SmartComputerLab");
  display.drawString(0, 14, "Topic");
  display.drawString(8, 26, top);
  display.drawString(0, 40, "Message");
  display.drawString(8, 52, mes);
  display.display();
}
```

```

void connect() {
  char cbuff[128];
  Serial.print("checking wifi...");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(1000);
  }
  Serial.print("\nconnecting...");
  while (!client.connect("IoT.GW5")) {
    Serial.print("."); delay(1000);
  }
  Serial.println("\nIoT.GW1 - connected!");
  client.subscribe("/esp32/test");
}

void messageReceived(String &topic, String &payload)
{
  MQTTpack rp; // received packet
  Serial.println("incoming: " + topic + " - " + payload);
  topic.toCharArray(rp.topic, 32); payload.toCharArray(rp.mess, 32);
  display_MQTT(rp.topic, rp.mess);
}

void setup() {
  Serial.begin(9600);
  Wire.begin(12, 14); // SCL, SDA
  Serial.println();
  display.init();
  display.flipScreenVertically();
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_16);
  display.drawString(0, 0, "Start MQTT");
  display.display();
  delay(2000);
  WiFi.begin(ssid, pass);
  client.begin(mqttServer, net);
  client.onMessage(messageReceived);
  connect();
}

int val=0;

void loop() {
  char vbuff[15];
  client.loop();
  delay(10); // <- fixes some issues with WiFi stability
  if (!client.connected()) { connect(); }
  sprintf(vbuff, "%d", val);
  client.publish("/esp32/test", vbuff);
  Serial.printf("published to /esp32/test:%s\n", vbuff);
  delay(6000); val++;
}

```

Il commence par l'inclusion des bibliothèques **WiFi.h** et **MQTT.h** et par les les déclarations des paramètres d'accès au **WiFi** et au broker **MQTT**.

Après la connexion de notre client MQTT – **IoT.GW1** au broker **MQTT** nous allons nous abonner (**subscribe**) aux plusieurs rubriques (**topics**).

Le **topic** de base est: **/esp32/test**

Dans la fonction d'initialisation **setup()** :

```

  WiFi.begin(ssid, pass);
  client.begin(mqttServer, net);
  client.onMessage(messageReceived);
  connect();

```



on initialise les paramètres (**credentials**) de WiFi et du client MQTT avec la fonction de **callback-messageReceived()** , puis on appelle la fonction **connect()** pour activer la connexion WiFi et s'abonner (**subscribe**) aux plusieurs **topic** et **sous-topic**.

Dans la boucle (tâche de fond) principale **loop()** , on vérifie la connexion **WiFi** et broker **MQTT**, puis on publie la valeur du compteur (val) sur le **topic** - **/esp32/test**

## A faire :

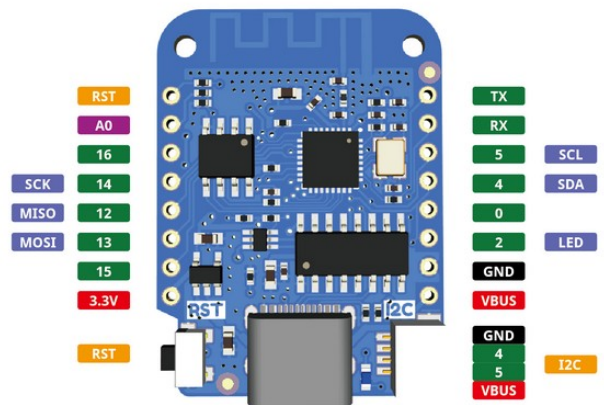
1. Charger l'application Android **MyMQTT** et la tester avec notre exemple.
2. Ajouter un capteur et envoyer les valeurs captées comme message MQTT
3. Séparer les fonctionnalités **publish** et **subscribe** sur **2 DevKits** : un DevKit envoie les message (publish) et l'autre les reçoit (subscribe) puis affiche sur son écran OLED.

## 2.2 Simple broker MQTT avec ESP8266

Le SoC **ESP8266** est le prédécesseur du SoC **ESP32**. Il contient un simple processeur Tensilica Xtensa LX106.

Un exemple de caractéristiques de ce SoC est indiqué ci-dessous.

- 32-bit [RISC](#) CPU: Tensilica Xtensa LX106, 80 MHz;
- 64 Kio de RAM instruction, 96 Kio de RAM data ;
- QSPI flash externe - 512 Kio à 4 Mio (supporte jusqu'à 16 Mio) ;
- [IEEE 802.11](#) b/g/n [Wi-Fi](#) ;
  - [TR switch](#) intégré, [balun](#), [LNA](#), amplificateur de puissance et [matching network](#) ;
  - Authentification par [WEP](#) ou [WPA/WPA2](#) ou bien réseau ouvert
- 16 broches [GPIO](#)
- Interfaces [SPI](#), [I2C](#);
- Interface [I<sup>2</sup>S](#) avec [DMA](#) (partageant les broches avec les GPIO) ;
- [UART](#) sur des broches dédiées, plus un UART dédié aux transmission pouvant être géré par GPIO2;
- 1 10-bit [ADC](#)



**Fig 2.1** Les broches d'un ESP8266-Wemos D1 mini

Voici le fichier `platformio.ini` pour initialiser l'utilisation d'une carte **Wemos d1\_mini\_lite** et importer la bibliothèque `uMQTTBroker.h`.

```
[env:d1_mini_lite]
platform = espressif8266
board = d1_mini_lite
framework = arduino
lib_deps = https://github.com/martin-ger/uMQTTBroker.git
```

Le code complet du mini broker inclut les bibliothèques `ESP8266WiFi.h` et `uMQTTBroker.h`.

Ensuite nous déclarons les credentials, le nom du point d'accès et le mot de passe.

Le broker peut fonctionner sur le réseau local associé à un point d'accès donné (`ssid`, `pass`), ou peut lui même devenir le point d'accès (`ssidAP`, `pasAP`).

La classe `myMQTTBroker` importée de la bibliothèque `uMQTTBroker.h` est **surchargée** par les méthodes de l'utilisateur, vous pouvez les modifier/compléter selon le besoin:

- `onConnect` – affichage de l'adresse IP
- `onDisconnect`
- `onAuth` – affichage du nom utilisateur et son mot de passe (optionnel)
- `onData` - affichage du **topic** et message associé

Les fonctions `startWiFiClient()` et `startWiFiAP()` permettent d'établir une connexion WiFi avec un point d'accès externe ou d'activer un propre point d'accès.

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
```

```

#include "uMQTTBroker.h"
#include <Wire.h> // Only needed for Arduino 1.6.5 and earlier
#include "SSD1306Wire.h" // legacy: #include "SSD1306.h"
SSD1306Wire display(0x3c, D2, D1); // SDA, SCL

float stab[2];
char ssid[] = "PhoneAP"; // your network SSID (name)
char pass[] = "smartcomputerlab"; // your network password
char ssidAP[] = "MyMQTT-6"; // softAP SSID (name)
char passAP[] = "smartcomputerlab"; // softAP network password
bool WiFiAP = true; // Do yo want the ESP as AP?

void display_SSD1306()
{
  char buff[32];
  display.init();
  display.flipScreenVertically();
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_16);
  display.drawString(0, 0, " MyMQTT-6 ");
  display.drawString(0, 24, " IP:192.168.5.1");
  display.setFont(ArialMT_Plain_10);
  display.drawString(8, 48, "SmartComputerLab");
  display.display();
}

class myMQTTBroker: public uMQTTBroker
{
public:
  virtual bool onConnect(IPAddress addr, uint16_t client_count)
  {
    Serial.println(addr.toString()+" connected");
    return true;
  }
  virtual void onDisconnect(IPAddress addr, String client_id)
  {
    Serial.println(addr.toString()+" (" +client_id+") disconnected");
  }
  virtual bool onAuth(String username, String password, String client_id)
  {
    Serial.println("Username/Password/ClientId: "+username+"/"+password+"/"+client_id);
    return true;
  }
  virtual void onData(String topic, const char *data, uint32_t length)
  {
    {
      char buf[24];
      char data_str[length+1];
      os_memcpy(data_str, data, length);
      data_str[length] = '\0';
      Serial.println("received topic '"+topic+"' with data '"+(String)data_str+"'");
      //printClients();
    }
  }

  // Sample for the usage of the client info methods
  virtual void printClients() {
    for (int i = 0; i < getClientCount(); i++)
    {
      IPAddress addr;
      String client_id;
      getClientAddr(i, addr);
      getClientId(i, client_id);
      Serial.println("Client "+client_id+" on addr: "+addr.toString());
    }
  }
};

myMQTTBroker myBroker;

void startWiFiClient()
{
  Serial.println("Connecting to "+(String)ssid);
  WiFi.mode(WIFI_STA);

```

```

WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: " + WiFi.localIP().toString());
}

void startWiFiAP()
{
    WiFi.mode(WIFI_AP);
    // softAP specific configuration
    if (!WiFi.softAPConfig(IPAddress(192,168,5,1), IPAddress(192,168,5,1), IPAddress(255,255,255,0)))
    {
        Serial.println("AP Config Failed");
    }
    WiFi.softAP(ssidAP, passAP);
    Serial.println("AP started");
    Serial.println("IP address: " + WiFi.softAPIP().toString());
}

void setup()
{
    Serial.begin(9600);
    Serial.println();
    Wire.begin(D1,D2); // SCL,SDA
    Serial.println();
    display_SSD1306();
    Serial.println();
    Serial.println("Initialized");
    // Start WiFi
    if (WiFiAP)
        startWiFiAP();
    else
        startWiFiClient();
    // Start the broker
    Serial.println("Starting MQTT broker");
    myBroker.init();
    // Subscribe to anything to monitor the received messages
    myBroker.subscribe("#");
}

int counter = 0;

void loop()
{
    // Publish the counter value as String
    //myBroker.publish("broker/counter", (String)counter++);
    // wait 10 seconds
    delay(10000);
}

```

Ci-dessous l'exemple d'exécution du broker en mode **softAP** avec l'adresse IP par défaut **192.168.4.1**

```

IP address: 192.168.4.1
Starting MQTT broker
received topic 'broker/counter' with data '0'
received topic 'broker/counter' with data '1'
received topic 'broker/counter' with data '2'
received topic 'broker/counter' with data '3'
received topic 'broker/counter' with data '4'
received topic 'broker/counter' with data '5'

```

## A faire :

1. Tester l'exemple du broker ci-dessus
2. Utiliser **2 DevKits pour tester notre broker**: un **DevKit** envoie les messages (publish) et l'autre les reçoit (subscribe) puis affiche sur son écran OLED.

# Laboratoire 3 – WiFi avec WiFiManager et serveur ThingSpeak.com (.fr)

## 3.1 Introduction

Dans ce laboratoire nous allons nous intéresser aux moyens de la **communication** et de la **présentation (stockage)** des résultats. Etant donné que le SoC ESP32 intègre les modems de WiFi et de Bluetooth nous allons étudier la communication par **WiFi**.

Principalement nous avons deux modes de fonctionnement **WiFi** – mode station **STA**, et mode point d'accès **softAP**. Dans le mode **STA** nous pouvons connecter notre carte à un point d'accès WiFi (qui peut être notre smartphone). Dans le mode **AP** nous créons un point d'accès sur la carte ESP32. Cet point d'accès peut être utilisé pour effectuer une configuration de la carte, par exemple en lui fournissant le nom du point d'accès (**ssid**) et le mot de passe pour ce point d'accès.

Un troisième mode appelé **ESP-NOW** permet de communiquer entre les cartes directement (sans un Point d'Accès), donc plus rapidement et à plus grande distance, par le biais des trames physiques **MAC**.

Une fois la communication WiFi est opérationnelle nous pouvons choisir un des multiples protocoles pour transmettre nos données : **UDP**, **TCP**, **HTTP/TCP**, .

Par exemple la carte peut fonctionner comme client ou serveur WEB.

### 3.1.1 Un programme de test – scrutation du réseau WiFi

Pour commencer notre étude de la communication WiFi essayons un exemple permettant de scruter notre environnement dans la recherche de point d'accès visibles par notre modem.

```
#include "WiFi.h"
#include <Wire.h>
#include "SSD1306Wire.h"
SSD1306Wire display(0x3c, 12, 14);

void display_SSD1306(char *text)
{
    char buff[256];
    strcpy(buff, text);
    display.init();
    display.flipScreenVertically();
    display.setTextAlignment(TEXT_ALIGN_LEFT);
    display.setFont(ArialMT_Plain_10);
    display.drawString(0, 0, "ETN - WiFi SCAN");
    display.drawString(0, 14, buff);
    display.display();
}

void setup()
{
    char buf[16];
    Serial.begin(9600);
    Wire.begin(12,14);
    Serial.println();
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(2000);
    Serial.println("Setup done");
}

void loop()
{
    int apmax=8, num;
    char buf[16];
    char tab[256];

    Serial.println("scan start");
    Serial.println("WiFi scan");
    // WiFi.scanNetworks will return the number of networks found
    int n = WiFi.scanNetworks();
    Serial.println("WiFi scan");
    if (n == 0) {
        Serial.println("no networks found");
    }
```

```

} else {
    //Serial.print(n);
    sprintf(buf, "WiFi AP#%d\n", n);
    Serial.println(buf);
    delay(2000);
    if(n<apmax) num=n; else num=apmax;
    //Serial.println(" networks found");
    for (int i = 0; i < num; ++i) {
        // Print SSID and RSSI for each network found
        Serial.print(i + 1);
        Serial.print(": ");
        WiFi.SSID(i).toCharArray(buf,16);
        Serial.print(buf);
        if(!i) strcpy(tab,buf);else strcat(tab,buf);strcat(tab, "\n");
        display_SSD1306(tab);
        //Serial.println(WiFi.SSID(i));
        Serial.print("  (");
        Serial.print(WiFi.RSSI(i));
        Serial.print(")");
        Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
        delay(10);
    }
}
Serial.println("");
delay(5000);
}

```

## A faire

Tester et analyser le programme ci-dessus.



## 3.2 Mode WiFi – STA, client WEB et serveur ThingSpeak

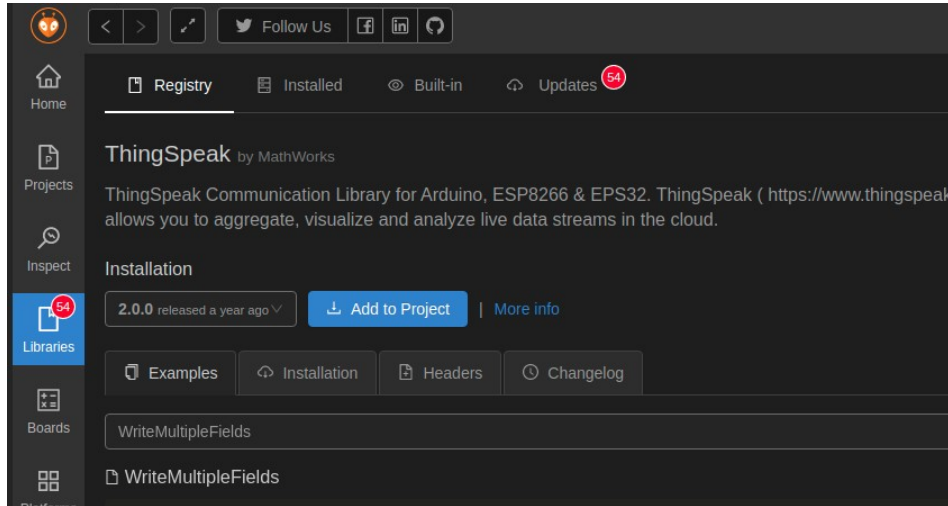
Une connexion WiFi permet de créer une application avec un client WEB. Ce client WEB peut envoyer les requêtes HTTP vers un serveur WEB.

Dans nos laboratoires nous allons utiliser un serveur IoT externe **ThingSpeak.com**.

**ThingSpeak** est un serveur «*open source*» qui peut être installé sur un PC ou sur une carte SBC.

Nous pouvons donc envoyer les requêtes HTTP (par exemple en format **GET** et les données attachées à **URL**) sur le serveur **ThingSpeak.com**.

La préparation de ces requêtes peut être laborieuse, heureusement il existe une bibliothèque **ThingSpeak.h** qui permet de simplifier cette tâche. Il faut donc installer la bibliothèque **ThingSpeak.h**.



Après l'inscription sur le serveur **ThingSpeak.com** vous créez un **channel** composé de max 8 **fields**. Ensuite vous pouvez envoyer et lire vos données dans ce **fields** à condition de fournir la clé d'écriture associée au **channel**.

Une écriture/envoi de plusieurs valeurs en virgule flottante est effectuée comme suit :

```
ThingSpeak.setField(1, sensor[0]); // préparation du field1
ThingSpeak.setField(2, sensor[1]); // préparation du field2
```

puis

```
ThingSpeak.writeFields(myChannelNumber[1], myWriteAPIKey[1]); // envoi
```

Voici le début d'un tel programme :

```
#include <WiFi.h>
#include "ThingSpeak.h"
char ssid[] = "PhoneAP"; // your network SSID (name)
char pass[] = "smartcomputerlab"; // your network passw
unsigned long myChannelNumber = 1;
const char * myWriteAPIKey="MEH7A0FHAMNWJE8P" ;
WiFiClient client;
void setup()
{
  Serial.begin(9600);
  WiFi.disconnect(true); // effacer de l'EEPROM WiFi credentials
  delay(1000);
  WiFi.begin(ssid, pass);
  delay(1000);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500); Serial.print(".");
  }
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip); Serial.println("WiFi setup ok");
  delay(1000);
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
}
```

Pour simplifier l'exemple dans la fonction de la boucle principale `loop()` nous allons envoyer les données d'un compteur.

```
int tout=10000; // en millisecondes
float luminosity=100.0, temperature=10.0;

void loop()
{
  ThingSpeak.setField(1, luminosity); // préparation du field1
  ThingSpeak.setField(2, temperature); // préparation du field1
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
  }
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  luminosity++;temperature++;
  delay(tout);
}
```

**Attention :** pour le serveur **ThingSpeak.com** utilisé gratuitement (max 1 an) la **valeur minimale** de `tout` est 20 secondes.



**Figure 2.1. ThingSpeak :** l'affichage des données envoyées sur le serveur type **ThingSpeak**

## A faire

1. Créer un canal **ThingSpeak** puis récupérer les paramètres du canal : *number* et *write key*.
2. Intégrer l'utilisation des capteurs de Température/Humidité et de la Luminosité.
3. Envoyer ces données cycliquement (par exemple toutes les 30 seconde) vers le serveur **ThingSpeak**.

### 3.3 Mode WiFi – STA et WiFiManager

La carte ESP32 permet de fonctionner en mode Point d'Accès qui permet de déployer un simple serveur WEB avec une page de configuration des paramètres sur la carte ESP32.

Parmi ces paramètres il y a la possibilité de proposer (et enregistrer) le nom d'un point d'accès et son mot de passe.

**WiFiManager** est une fonction qui réalise ce type d'opérations.

Voici le contenu de `platformio.ini` avec la bibliothèque **WiFiManager**

```
[env:lolin_d32]
platform = espressif32
board = lolin_d32
framework = arduino
lib_deps =
https://github.com/tzapu/WiFiManager.git
```

et le programme de test pour l'utilisation du **WiFiManager**.

```
#include <WiFiManager.h> // https://github.com/tzapu/WiFiManager

void setup() {
    WiFi.mode(WIFI_STA); // set mode to STA+AP
    Serial.begin(9600);
    WiFiManager wm;
    //reset settings - wipe credentials for testing
    wm.resetSettings(); // à tester , puis à commenter
    // Automatically connect using saved credentials,
    bool res;
    res = wm.autoConnect("ESP32AP",NULL); // no password
    if(!res) {
        Serial.println("Failed to connect"); // ESP.restart();
    }
    else {
        //if you get here you have connected to the WiFi
        Serial.println("connected...yeey :)");
    }
}

void loop() { }
```

Sur votre terminal IDE **WiFiManager** affiche ses paramètres et l'état de fonctionnement.

Vous devrez vous connecter avec votre smartphone sur son pont d'accès (ici) ESP32AP et aller avec votre navigateur sur 192.168.4.1 pour accéder à la page d'accueil , puis choisir votre point d'accès local et fournir le mot de passe.

Le point d'accès et le mot de passe seront enregistrés sur votre carte.

N'oubliez pas de **commenter** la ligne :

```
wm.resetSettings();
```

pour pouvoir garder ces valeurs en mémoire **EEPROM** du ESP32.

#### Attention:

Sur certaines cartes ESP32 il faut effacer intégralement la mémoire flash avant le chargement du code avec l'utilitaire **esptool.py**:

```
~/platformio/packages/framework-espidf/components/esptool_py/esptool/esptool.py
~/arduino-1.8.1$ ./hardware/espressif/esp32/tools/esptool.py -p /dev/ttyUSB0 -b 460800 erase_flash
```

Voici le déroulement d'affichage de **WiFiManager** dans le cas de connexion sur un nouveau point d'accès.

```
*WM: [3] WIFI station disconnect
*WM: [3] WiFi station enable
*WM: [2] Disabling STA
```

```

*WM: [2] Enabling AP
*WM: [1] StartAP with SSID:  ESP32AP
*WM: [2] AP has anonymous access!
*WM: [1] SoftAP Configuration
*WM: [1] -----
*WM: [1] ssid:                ESP32AP
*WM: [1] password:
*WM: [1] ssid_len:            7
*WM: [1] channel:            1
*WM: [1] authmode:
*WM: [1] ssid_hidden:
*WM: [1] max_connection:     4
*WM: [1] country:            CN
*WM: [1] beacon_interval:    100(ms)
*WM: [1] -----
*WM: [1] AP IP address: 192.168.4.1
*WM: [3] setupConfigPortal
*WM: [1] Starting Web Portal
*WM: [3] dns server started with ip:  192.168.4.1
*WM: [2] HTTP server started
*WM: [2] WiFi Scan completed in 5509 ms
*WM: [2] Config Portal Running, blocking, waiting for clients...

*WM: [3] WIFI station disconnect
*WM: [3] WiFi station enable
*WM: [2] Disabling STA
*WM: [2] Enabling AP
*WM: [1] StartAP with SSID:  ESP32AP
*WM: [2] AP has anonymous access!
*WM: [1] SoftAP Configuration
*WM: [1] -----
*WM: [1] ssid:                ESP32AP
*WM: [1] password:
*WM: [1] ssid_len:            7
*WM: [1] channel:            1
*WM: [1] authmode:
*WM: [1] ssid_hidden:
*WM: [1] max_connection:     4
*WM: [1] country:            CN
*WM: [1] beacon_interval:    100(ms)
*WM: [1] -----
*WM: [1] AP IP address: 192.168.4.1
*WM: [3] setupConfigPortal
*WM: [1] Starting Web Portal
*WM: [3] dns server started with ip:  192.168.4.1
*WM: [2] HTTP server started
*WM: [2] WiFi Scan completed in 5509 ms
*WM: [2] Config Portal Running, blocking, waiting for clients...

WM: [2] NUM CLIENTS: 0
*WM: [3] -> connectivitycheck.platform.hicloud.com
*WM: [2] <- Request redirected to captive portal
*WM: [2] NUM CLIENTS: 1
*WM: [2] <- HTTP Root
*WM: [3] -> 192.168.4.1
*WM: [3] lastconxresulttmp: WL_IDLE_STATUS
*WM: [3] lastconxresult: WL_DISCONNECTED
*WM: [2] WiFi Scan completed in 5106 ms
*WM: [2] <- HTTP Wifi
*WM: [2] Scan is cached 7 ms ago
*WM: [1] 18 networks found
*WM: [2] DUP AP: FreeWifi_secure
*WM: [2] DUP AP: SFR WiFi Mobile
*WM: [2] DUP AP: Livebox-C82A
*WM: [2] AP: -52 DIRECT-G8M2070 Series
*WM: [2] AP: -60 orange
*WM: [2] AP: -62 Livebox-08B0
*WM: [2] AP: -63 VAIO-MQ35AL
*WM: [2] AP: -75 Livebox-08B0_Ext
*WM: [2] AP: -84 FreeWifi_secure
*WM: [2] AP: -85 Livebox-3D36
*WM: [2] AP: -88 SFR WiFi Mobile
*WM: [2] AP: -90 SFR WiFi FON

```

```

*WM: [2] AP: -93 Livebox-F936
*WM: [2] AP: -93 Livebox-C82A
*WM: [2] AP: -94 SFR_E0B0
*WM: [2] AP: -95 SFR_A0A0
*WM: [2] AP: -95 freebox_ODTMTH
*WM: [2] AP: -96 Bbox-EA1EB632
*WM: [3] lastconxresulttmp: WL_IDLE_STATUS
*WM: [3] lastconxresult: WL_DISCONNECTED
*WM: [3] Sent config page
*WM: [3] -> connectivitycheck.platform.hicloud.com
*WM: [2] <- Request redirected to captive portal
*WM: [2] <- HTTP WiFi save
*WM: [3] Method: POST
*WM: [3] Sent wifi save page
*WM: [2] processing save
*WM: [2] Connecting as wifi client...
*WM: [3] STA static IP:
*WM: [2] setSTAConfig static ip not set, skipping
*WM: [1] CONNECTED:
*WM: [1] Connecting to NEW AP: Livebox-08B0
*WM: [3] Using Password: G79ji6dtEptVTPWmZP
*WM: [3] WiFi station enable
*WM: [1] connectTimeout not set, ESP waitForResult...
*WM: [2] Connection result: WL_CONNECTED
*WM: [3] lastconxresult: WL_CONNECTED
*WM: [1] Connect to new AP [SUCCESS]
*WM: [1] Got IP Address:
*WM: [1] 192.168.1.20
*WM: [2] disconnect configportal
dhcps: send_nak>>udp_sendto result 0
*WM: [2] restoring usermode STA
*WM: [2] wifi status: WL_CONNECTED
*WM: [2] wifi mode: STA
*WM: [1] config portal exiting
connected...yeey :)
*WM: [3] unloading

```

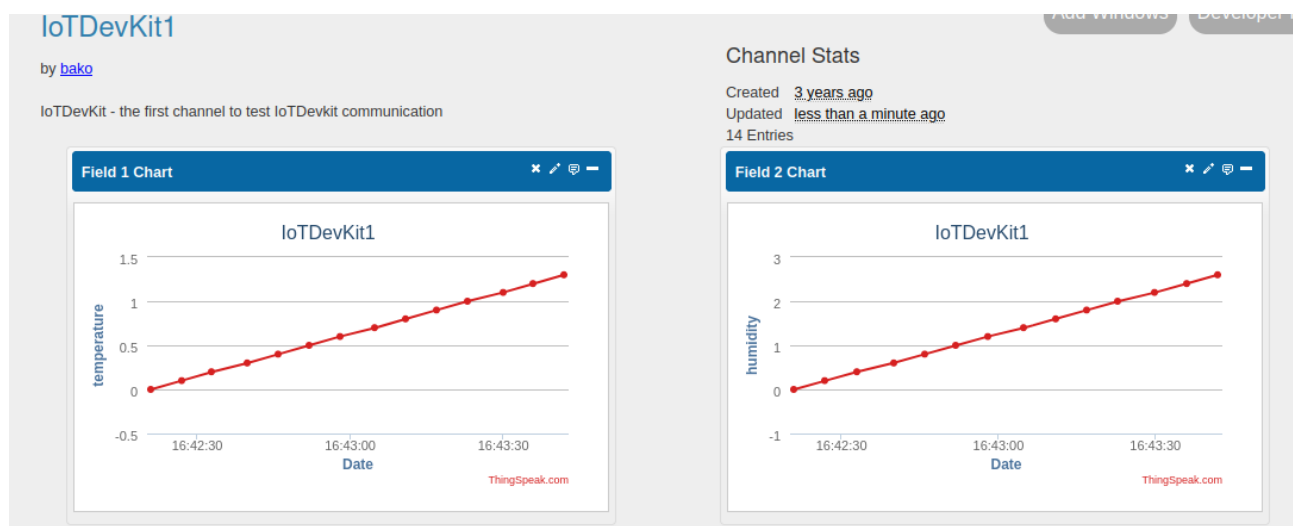
### 3.4 Envoi des données sur ThingSpeak avec WiFiManager

```
#include <Arduino.h>
#include <WebServer.h>
#include <WiFiManager.h>
#include "ThingSpeak.h"
char ssid[] = "Livebox-08B0"; // your network SSID (name)
char pass[] = "G79ji6dtEptVTPWmZP"; // your network passw
unsigned long myChannelNumber = 1538804;
const char * myWriteAPIKey="YOX31M0EDKO0JATK" ;
int dout=15;
WiFiClient client;

void setup() {
  WiFi.mode(WIFI_STA);
  Serial.begin(9600);
  WiFiManager wm;
  //wm.resetSettings();
  bool res;
  res = wm.autoConnect("ESP32AP",NULL); // password protected ap
  if(!res) {
    Serial.println("Failed to connect"); // ESP.restart();
  }
  else {
    //if you get here you have connected to the WiFi
    Serial.println("connected...yeey :)");
  }
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
}

float temperature=0.0,humidity=0.0;

void loop() {
  Serial.println("Fields update");
  ThingSpeak.setField(1, temperature);
  ThingSpeak.setField(2, humidity);
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(dout*1000);
  temperature+=0.1;
  humidity+=0.2;
}
```





## 3.5 Réception des données de ThingSpeak

```
#include <Arduino.h>
#include <WebServer.h>
#include <WiFiManager.h>
#include "ThingSpeak.h"
char ssid[] = "Livebox-08B0"; // your network SSID (name)
char pass[] = "G79ji6dtEptVTPWmZP"; // your network passw
unsigned long myChannelNumber = 1538804;
const char * myWriteAPIKey="YOX31M0EDKO0JATK" ;

WiFiClient client;

void setup() {
  WiFi.mode(WIFI_STA);
  Serial.begin(9600);
  WiFiManager wm;
  //reset settings - wipe credentials for testing
  // wm.resetSettings();
  bool res;
  res = wm.autoConnect("ESP32AP",NULL); // password protected ap
  if(!res) {
    Serial.println("Failed to connect");
    // ESP.restart();
  }
  else {
    //if you get here you have connected to the WiFi
    Serial.println("connected...yeey :)");
  }
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
}

float temperature=0.0,humidity=0.0;
float tem,hum;

void loop() {
  Serial.println("Fields update");
  ThingSpeak.setField(1, temperature);
  ThingSpeak.setField(2, humidity);
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(6000);
  temperature+=0.1;
  humidity+=0.2;
  tem =ThingSpeak.readFloatField(myChannelNumber,1);
  // if channel is private you need to add the red key: AVG5MID7I5SPHIK8
  tem =ThingSpeak.readFloatField(myChannelNumber,1,"20E9AQVFW7Z6XXOM");
  Serial.print("Last temperature:");
  Serial.println(tem);
  delay(6000);
  hum =ThingSpeak.readFloatField(myChannelNumber,2,"20E9AQVFW7Z6XXOM");
  Serial.print("Last humidity:");
  Serial.println(hum);
  delay(15000);
}
```

Voici l'affiche correspondant à l'exécution de ces programme :

```
*WM: [3] STA static IP:
*WM: [2] setSTAConfig static ip not set, skipping
*WM: [1] Connecting to SAVED AP: Livebox-08B0
*WM: [3] Using Password: G79ji6dtEptVTPWmZP
*WM: [3] WiFi station enable
*WM: [1] connectTimeout not set, ESP waitForResult...
*WM: [2] Connection result: WL_CONNECTED
*WM: [3] lastconxresult: WL_CONNECTED
*WM: [1] AutoConnect: SUCCESS
*WM: [1] STA IP Address: 192.168.1.20
connected...yeey :)
```

```
ThingSpeak begin
*WM: [3] unloading
Fields update
Last temperature:0.00
Last humidity:0.00
Fields update
Last temperature:0.10
Last humidity:0.20
Fields update
Last temperature:0.20
Last humidity:0.40
Fields update
Last temperature:0.30
Last humidity:0.60
Fields update
Last temperature:0.40
```

### A faire (comme dans l'exemple précédent) :

1. Créer un canal **ThingSpeak** puis récupérer les paramètres du canal : *number* , *read key* et *write key*.
  2. Intégrer l'utilisation des capteurs de Température/Humidité et de la Luminosité.
- Envoyer ces données cycliquement (par exemple toutes les 30 seconde) vers le serveur **ThingSpeak**.

# Laboratoire 4 – communication longue distance avec LoRa (Long Range)

## 4.1 Introduction

Dans ce laboratoire nous allons nous intéresser à la technologie de transmission **Long Range** essentielle pour la communication entre les objets.

Longe Range ou **LoRa** permet de transmettre les données à la distance d'un kilomètre ou plus avec les débits allant de quelques centaines de bits par seconde aux quelques dizaines de kilobits (100bit – 75Kbit).

### 4.1.1 Modulation LoRa

Modulation LoRa a trois paramètres :

- **freq** – *frequency* ou fréquence de la porteuse de 868 à 870 MHz,
- **sf** – *spreading factor* ou étalement du spectre ou le nombre de modulations par un bit envoyé (64-4096 exprimé en puissances de 2 – 7 à 12)
- **sb** – *signal bandwidth* ou bande passante du signal (31250 Hz à 500KHz)

Par défaut on utilise : **freq**=434MHz, **sf**=7, et **sb**=125KHz

Notre DevKit peut être complété par une carte d'extension – modem LoRa.

La paramétrisation du modem LoRa est facilité par la bibliothèque **LoRa.h** à inclure dans vos programmes.

```
#include <LoRa.h>
```

#### 4.1.1.1 Fréquence LoRa en France

```
LoRa.setFrequency(434E6);
```

définit la **fréquence** du modem sur **434 MHz**.

#### 4.1.1.2 Facteur d'étalement en puissance de 2

Le **facteur d'étalement (sf)** pour la modulation LoRa varie de **2<sup>7</sup>** à **2<sup>12</sup>** Il indique combien de **chirps** sont utilisés pour transporter un bit d'information.

```
LoRa.setSpreadingFactor(8);
```

définit le facteur d'étalement à 10 ou 1024 signaux par bit.

#### 4.1.1.3 Bande passante

La **largeur de bande de signal (sb)** pour la modulation LoRa varie de **31,25 kHz** à **500 kHz** (31,25, 62,5, 125,0, 250,0, 500,0). Plus la bande passante du signal est élevée, plus le débit est élevé.

```
LoRa.setSignalBandwidth (125E3);
```

définit la largeur de bande du signal à 125 KHz.

## 4.1.2 Paquets LoRa

La **classe LoRa** est activée avec le constructeur **begin()** qui prend éventuellement un argument, la fréquence.

```
int begin (longue freq);
```

Pour créer un paquet LoRa, nous commençons par:

```
int beginPacket();
```

pour terminer la construction du paquet que nous utilisons

```
int endPacket ();
```

Les données sont **envoyées** par les fonctions:

```
virtual size_t write (uint8_t byte);  
virtual size_t write (const uint8_t * buffer, taille_taille);
```

Dans notre cas, nous utilisons fréquemment la deuxième fonction avec le tampon contenant 4 données en virgule flottante.

La réception **en continue** des paquets LoRa peut être effectuée via la méthode **parsePacket ()** .

```
int parsePacket();
```

Si le paquet est présent dans le tampon de réception du modem LoRa, cette méthode renvoie une valeur entière égale à la taille du paquet (charge utile de trame).

Pour lire efficacement le paquet du tampon, nous utilisons les méthodes :

```
LoRa.available() et LoRa.read () .
```

Les paquets LoRa sont envoyés comme chaînes des octets. Ces chaînes doivent porter différents types de données.

Afin d'accommoder ces différents formats nous utilisons les **union**.

Par exemple pour formater un paquet avec 4 valeurs en virgule flottante nous définissons une union type :

```
union pack  
{  
    uint8_t frame[16]; // trames avec octets  
    float data[4]; // 4 valeurs en virgule flottante  
} sdp ; // paquet d'émission
```

Pour les paquets plus évolués nous avons besoin d'ajouter les en-têtes . Voici un exemple d'une union avec les paquets structurés.

```
union tspack  
{  
    uint8_t frame[48];  
    struct packet  
    {  
        uint8_t head[4]; uint16_t num; uint16_t tout; float sensor[4];  
    } pack;  
} sdf,sbf,rdf,rbf; // data frame and beacon frame
```

## 4.2 Premier exemple – émetteur et récepteur des paquets LoRa



Figure 4.1 Un lien LoRa avec un émetteur et un récepteur



Figure 4.2 Module LoRa avec une connexion sur bus SPI

La partie initiale du code est la même pour l'émetteur et pour le récepteur. Dans cette partie nous insérons les bibliothèque de connexion par le bus SPI (`SPI.h`) et de communication avec le modem LoRa (`LoRa.h`). Nous proposons les paramètres par défaut pour le lien radio LoRa.

```
#include <SPI.h>
#include <LoRa.h>
#define SCK      18    // GPIO18 -- SX127x's SCK
#define MISO     19    // GPIO19 -- SX127x's MISO
#define MOSI     23    // GPIO23 -- SX127x's MOSI
#define SS       5     // GPIO05 -- SX127x's CS
#define RST      15    // GPIO15 -- SX127x's RESET
#define DIO      26    // GPIO26 -- SX127x's IRQ (Interrupt Request)
#define freq     434E6
#define sf       7
#define sb       125E3
```

Nous définissons également le format d'un paquet dans la trame LoRa avec 4 champs `float` pour les données des capteurs.

```
union pack
{
    uint8_t frame[16]; // trames avec octets
    float data[4];     // 4 valeurs en virgule flottante
} sdp ; // paquet d'émission
```

Dans la fonction `setup()` nous initialisons les connexions avec le modem LoRa et les paramètres radio.

```
void setup() {
    Serial.begin(9600);
    SPI.begin(SCK, MISO, MOSI, SS);
    LoRa.setPins(SS, RST, DIO);
    Serial.println(); delay(100); Serial.println();
    if (!LoRa.begin(freq)) {
        Serial.println("Starting LoRa failed!");
        while (1);
    }
    Serial.println("Starting LoRa OK!");
    LoRa.setSpreadingFactor(sf);
    LoRa.setSignalBandwidth(sb);
}
```

Enfin dans la fonction `loop()` nous préparons les données à envoyer dans le paquet LoRa de façon cyclique , une fois toutes les 2 secondes.

```
float d1=0.0, d2=0.0 ;

void loop() // la boucle de l'émetteur
{
  Serial.print("New Packet:") ;
  LoRa.beginPacket(); // start packet
  sdp.data[0]=d1;
  sdp.data[1]=d2;
  LoRa.write(sdp.frame,16);
  LoRa.endPacket();
  Serial.printf("%2.2f,%2.2f\n",d1,d2);
  d1=d1+0.1; d2=d2+0.2;
  delay(2000);
}
```

Le programme du récepteur contient les mêmes déclarations et la fonction de `setup()` que le programme de l'émetteur. Le code ci dessous illustre seulement la partie différente.

```
#include <Arduino.h>

#include <SPI.h>
#include <LoRa.h>
#define SCK 18 // GPIO18 -- SX127x's SCK
#define MISO 19 // GPIO19 -- SX127x's MISO
#define MOSI 23 // GPIO23 -- SX127x's MOSI
#define SS 5 // GPIO05 -- SX127x's CS
#define RST 15 // GPIO15 -- SX127x's RESET
#define DIO 26 // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq 434E6
#define sf 7
#define sb 125E3
union pack
{
  uint8_t frame[16]; // trames avec octets
  float data[4]; // 4 valeurs en virgule flottante
} rdp ; // paquet de réception

void setup()
{
  Serial.begin(9600);
  SPI.begin(SCK,MISO,MOSI,SS);
  LoRa.setPins(SS,RST,DIO);
  Serial.println();delay(100);Serial.println();
  if (!LoRa.begin(freq)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  Serial.println("Starting LoRa OK!");
  LoRa.setSpreadingFactor(sf);
  LoRa.setSignalBandwidth(sb);
}

float d1=0.0, d2=0.0 ;
int rssi;

void loop()
{
  int packetLen;
  packetLen=LoRa.parsePacket();
  if(packetLen==16)
  {
    int i=0;
    while (LoRa.available()) {
      rdp.frame[i]=LoRa.read();i++;
    }
  }
```



```

d1=rdp.data[0];d2=rdp.data[1];
rssi=LoRa.packetRssi(); // force du signal en réception en dB
Serial.printf("Received packet:%2.2f,%2.2f\n",d1,d2);
Serial.printf("RSSI=%d\n", rssi);
}
}

```

### A faire :

1. Tester le code ci-dessus et analyser la force du signal en réception. Par exemple **-60 dB** signifie que le signal reçu est  $10^6$  plus faible que le signal d'émission.
2. Modifier les paramètres LoRa par exemple **freq=435E6**, **sf=10**, **sb=250E3** et tester le résultats de transmission.
3. Afficher les données envoyées/reçues sur l'écran OLED

## 4.3 onReceive() – récepteur des paquets LoRa avec une interruption

Les interruptions permettent de ne pas exécuter en boucle l'opération d'attente d'une nouvelle trame dans le tampon du récepteur. Une fonction-tâche séparée est réveillée automatiquement après l'arrivée d'une nouvelle trame.

Cette fonction, souvent appelée `onReceive()` doit être marquée dans le `setup()` par :

```
#include <Arduino.h>

#include <SPI.h>
#include <LoRa.h>
#define SCK 18 // GPIO18 -- SX127x's SCK
#define MISO 19 // GPIO19 -- SX127x's MISO
#define MOSI 23 // GPIO23 -- SX127x's MOSI
#define SS 5 // GPIO05 -- SX127x's CS
#define RST 15 // GPIO15 -- SX127x's RESET
#define DI0 26 // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq 434E6
#define sf 7
#define sb 125E3
```

La fonction **ISR** (*Interrupt Service Routine*) ci-dessous permet de **capter** une nouvelle trame.

```
void onReceive(int packetSize)
{
  int rssi=0;
  union pack
  {
    uint8_t frame[16]; // trames avec octets
    float data[4]; // 4 valeurs en virgule flottante
  } rdp ; // paquet de réception
  if (packetSize == 0) return; // if there's no packet, return
  int i=0;
  if (packetSize==16)
  {
    while (LoRa.available())
    {
      rdp.frame[i]=LoRa.read();i++;
    }
    rssi=LoRa.packetRssi();
    Serial.printf("Received packet:%2.2f,%2.2f\n",rdp.data[0],rdp.data[1]);
    Serial.printf("RSSI=%d\n",rssi);
  }
}

void setup() {
  Serial.begin(9600);
  SPI.begin(SCK,MISO,MOSI,SS);
  LoRa.setPins(SS,RST,DI0);
  Serial.println();delay(100);Serial.println();
  if (!LoRa.begin(freq)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  Serial.println("Starting LoRa OK!");
  LoRa.setSpreadingFactor(sf);
  LoRa.setSignalBandwidth(sb);
  LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
  LoRa.receive(); // pour activer l'interruption (une fois)
  // puis après chaque émission
}

void loop()
{
  Serial.println("in the loop") ;
  delay(5000) ;
}
```

L'affichage pendant l'exécution :

```

Received packet:90.90,181.80
RSSI=-52
Received packet:91.00,182.00
RSSI=-52
Received packet:91.10,182.20
RSSI=-53
in the loop
Received packet:91.20,182.40
RSSI=-53
Received packet:91.30,182.60
RSSI=-53
in the loop

```

## A faire :

1. Tester le code ci-dessus.
2. Afficher les données reçues sur l'écran OLED
3. Transférer les données reçues dans les variables externes (problème?).
4. La solution consiste à utiliser une file de message - **queue**

```

#include <Arduino.h>

..
static QueueHandle_t msg_queue;
union pack
{
  uint8_t frame[16]; // trames avec octets
  float data[4]; // 4 valeurs en virgule flottante
} rdp ; // paquet de réception

void onReceive(int packetSize)
{
  int rssi=0;
  uint8_t buff[16];
  if (packetSize == 0) return; // if there's no packet, return
  int i=0;
  if (packetSize==16) {
    while (LoRa.available()) { buff[i]=LoRa.read();i++; }
    xQueueReset(msg_queue);
    xQueueSend(msg_queue, (void *)buff, 16);
  }
}

void setup() {
  Serial.begin(9600);
  SPI.begin(SCK,MISO,MOSI,SS);
  LoRa.setPins(SS,RST,DIO);
  Serial.println();delay(100);Serial.println();
  if (!LoRa.begin(freq)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  Serial.println("Starting LoRa OK!");
  msg_queue = xQueueCreate(4, sizeof(rdp)); // 4 éléments type union rdp
  LoRa.setSpreadingFactor(sf);
  LoRa.setSignalBandwidth(sb);
  LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
  LoRa.receive(); // pour activer l'interruption (une fois)
  // puis après chaque émission
}

void loop()
{
  xQueueReceive(msg_queue, (void *)rdp.frame, 1000);
  Serial.printf("Received packet:%2.2f,%2.2f\n", rdp.data[0], rdp.data[1]);
}

```

## Laboratoire 5 - Développement de simples passerelles IoT

Dans ce laboratoire nous allons développer une architecture intégrant plusieurs dispositifs essentiels pour la création d'un système IoT complet. Le dispositif central sera la passerelle (*gateway*) entre les liens LoRa et la communication par WiFi.

### 5.1 Passerelle LoRa-ThingSpeak

La passerelle permettra de retransmettre les données reçues sur un lien **LoRa** et de les envoyer par sur une connexion WiFi vers un serveur **ThingSpeak**

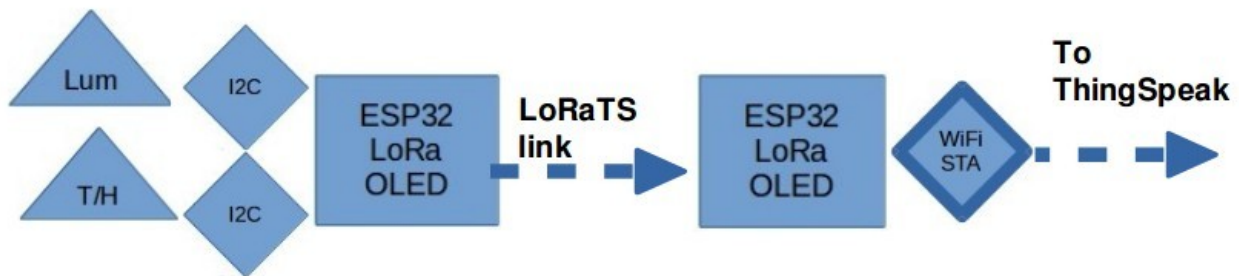


Figure 4.1 Une simple architecture IoT avec un terminal à 2 capteurs et une passerelle **LoRa-WiFi**

#### 5.1.1 Le principe de fonctionnement

La passerelle attend les messages LoRa envoyés dans le format prédéfini **LoRaTS** sur une interruption I/O redirigée vers la fonction (ISR) `onReceive()`. Elle les stocke dans une file de messages (*queue*) en gardant seulement le dernier paquet. La tâche principale, dans la boucle `loop()` récupère ce paquet dans la file par la fonction `xQueueReceive()` puis l'envoie sur le serveur **ThingSpeak**.

Le contenu d'un paquet en format **LoRaTS** est transformé en un ou plusieurs fonctions :

`ThingSpeak.setField(fn,value) ;`

et la fonction

`ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);`

#### 5.1.2 Les éléments du code

Ci-dessous nous présentons les éléments du code de la passerelle. Les messages LoRa sont envoyés dans un **format simplifié** de **LoRaTS**. Nous avons défini le type de l'union/structure d'un paquet (`pack_t`) de la façon suivante. Le format simplifié correspond au transfert des paquets LoRa dans le laboratoire précédent.

```
typedef union          // format simplifie
{
    uint8_t frame[16]; // trames avec octets
    float  data[4];    // 4 valeurs en virgule flottante
} pack_t ; // paquet d'émission

typedef union // format de base
{
    uint8_t frame[40];
    struct
    {
        uint8_t head[4]; // packet header
        int chnum;       // channel number
        char key[16];    // write or read key
        float sensor[4];
    } pack;
} pack_t;
```

Pour le format de base dans l'en-tête **head[4]** nous indiquons :

- head[0]** – adresse de destination, **0x00** – passerelle, **0xff** – diffusion
- head[1]** – adresse de source
- head[2]** – type du message – data **write/read**, **control**, ..
- head[3]** – le masque **0xC0** signifie **field1** et **field2**

Les paquets qui arrivent par le lien LoRa sont captés par l'ISR **onReceive()** et déposés dans une file d'attente **FreeRTOS**. Cette file doit être déclarée par :

```
QueueHandle_t dqueue; // queues for data packets
```

Puis, dans le **setup()** on instancie la file en lui fournissant le nombre d'éléments et la taille d'un élément:

```
dqueue = xQueueCreate(4,16); // queue for 4 data packets in simple format

void onReceive(int packetSize)
{
    pack_t rbuff; // receive buffer with pack_t format
    int i=0;
    if (packetSize == 0) return; // if there's no packet, return
    i=0;
    if (packetSize==16) // 16 pour le format simplifie , 40 format de base
    {
        while (LoRa.available()) {
            rbuff.frame=LoRa.read();i++;
        }
        xQueueReset(dqueue); // to keep only the last element
        xQueueSend(dqueue, rbuff, portMAX_DELAY);
    }
    delay(200);
}
```

La fonction de **setup()** :

```
void setup()
{
    Serial.begin(9600);
    WiFi.mode(WIFI_STA);
    WiFiManager wm;
    // wm.resetSettings();
    bool res;
    res = wm.autoConnect("ESP32AP",NULL); // password protected ap
    if(!res)
    {
        Serial.println("Failed to connect");// ESP.restart();
    }
    else
    {
        //if you get here you have connected to the WiFi
        Serial.println("connected...yeey :)");
    }
    ThingSpeak.begin(client); // connexion (TCP) du client au serveur
    delay(1000);
    Serial.println("ThingSpeak begin");
    Serial.println("Start Lora");
    SPI.begin(SCK,MISO,MOSI,SS);
    LoRa.setPins(SS,RST,DIO);
    Serial.println();delay(100);Serial.println();
    if (!LoRa.begin(freq)) {
        Serial.println("Starting LoRa failed!");
        while (1);
    }
    Serial.println("Starting LoRa OK!");
    LoRa.setSpreadingFactor(sf);
    LoRa.setSignalBandwidth(sb);
    dqueue = xQueueCreate(4,16); // queue for 4 simple data packets
    LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
    LoRa.receive(); // pour activer l'interruption (une fois)
}
```

La fonction de la tâche en `loop()` :

Pour le format simplifié :

```
void loop()
{
  pack_t rp;    // packet elements to send
  xQueueReceive(dqueue, rp.frame, portMAX_DELAY); // 6s, default: portMAX_DELAY
  ThingSpeak.setField(1, rp.data[0]);
  ThingSpeak.setField(2, rp.data[1]);
  ThingSpeak.setField(3, rp.data[2]);
  ThingSpeak.setField(4, rp.data[3]);
  Serial.printf("d1=%2.2f, d2=%2.2f, d3=%2.2f, d4=%2.2f\n",
    rp.data[0], rp.data[1], rp.data[2], rp.data[3]);
  while (WiFi.status() != WL_CONNECTED) { delay(500); }
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(mindel); // mindel is the min waiting time before sending to ThingSpeak
  //LoRa.receive();
}
```

Pour le format de base :

```
loop()
{
  pack_t sb;    // packet elements to send
  xQueueReceive(dqueue, sb.frame, portMAX_DELAY); // 6s, default: portMAX_DELAY
  if (sb.pack.head[1] == 0x01 || sb.pack.head[1] == 0x00)
  {
    if (sb.pack.head[2] & 0x80) ThingSpeak.setField(1, sb.pack.sensor[0]);
    if (sb.pack.head[2] & 0x40) ThingSpeak.setField(2, sb.pack.sensor[1]);
    if (sb.pack.head[2] & 0x20) ThingSpeak.setField(3, sb.pack.sensor[2]);
    if (sb.pack.head[2] & 0x10) ThingSpeak.setField(4, sb.pack.sensor[3]);
    while (WiFi.status() != WL_CONNECTED) { delay(500); }
    ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  }
  LoRa.receive();
}
```

### 5.1.3 Code complet pour une passerelle des paquets en format de base

```
#include <Arduino.h>
#include <WebServer.h>
#include <WiFiManager.h>
#include <SPI.h>
#include <LoRa.h>
#include "ThingSpeak.h"
char ssid[] = "Livebox-08B0"; // your network SSID (name)
char pass[] = "G79ji6dtEptVTPWmZP"; // your network passw
unsigned long myChannelNumber = 1538804;
const char * myWriteAPIKey="YOX31M0EDKO0JATK" ;
int dout=15;
WiFiClient client;

#define SCK 18 // GPIO18 -- SX127x's SCK
#define MISO 19 // GPIO19 -- SX127x's MISO
#define MOSI 23 // GPIO23 -- SX127x's MOSI
#define SS 5 // GPIO05 -- SX127x's CS
#define RST 15 // GPIO15 -- SX127x's RESET
#define DI0 26 // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq 434E6
#define sf 7
#define sb 125E3

union
{
  uint8_t frame[16]; // trames avec octets
  float data[4]; // 4 valeurs en virgule flottante
} rdp ; // paquet de reception

QueueHandle_t msg_queue; // queues for data packets

void onReceive(int packetSize)
{
  int rssi=0;
  uint8_t buff[16];
  if (packetSize == 0) return; // if there's no packet, return
  int i=0;
  if (packetSize==16)
  {
    while (LoRa.available()) { buff[i]=LoRa.read();i++; }
    xQueueReset(msg_queue);
    xQueueSend(msg_queue, (void *)buff, 16);
  }
}

void setup()
{
  WiFi.mode(WIFI_STA);
  Serial.begin(9600);
  WiFiManager wm;
  //wm.resetSettings();
  bool res;
  res = wm.autoConnect("ESP32AP",NULL); // password protected ap
  if(!res) {
    Serial.println("Failed to connect"); // ESP.restart();
  }
  else {
    //if you get here you have connected to the WiFi
    Serial.println("connected...yeey :)");
  }
  msg_queue = xQueueCreate(4,16); // queue for 4 data packets in simple format
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");

  SPI.begin(SCK,MISO,MOSI,SS);
  LoRa.setPins(SS,RST,DI0);
  Serial.println();delay(100);Serial.println();
```



```

if (!LoRa.begin(freq)) {
  Serial.println("Starting LoRa failed!");
  while (1);
}
Serial.println("Starting LoRa OK!");
LoRa.setSpreadingFactor(sf);
LoRa.setSignalBandwidth(sb);
LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
LoRa.receive(); // pour activer l'interruption (une fois)
// puis après chaque émission
}

void loop()
{
  // packet elements to send
  xQueueReceive(msg_queue, (void *)rdp.frame, 6000); // 6s, default:portMAX_DELAY
  //LoRa.sleep();
  ThingSpeak.setField(1, rdp.data[0]);
  ThingSpeak.setField(2, rdp.data[1]);
  ThingSpeak.setField(3, rdp.data[2]);
  ThingSpeak.setField(4, rdp.data[3]);
  Serial.println("in the loop");
  Serial.printf("d1=%2.2f, d2=%2.2f, d3=%2.2f, d4=%2.2f\n",
    rdp.data[0], rdp.data[1], rdp.data[2], rdp.data[3]);
  while (WiFi.status() != WL_CONNECTED) { delay(500); }
  ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
  delay(15000); // 15 sec min pour ThingSpeak.com
  //LoRa.receive();
}

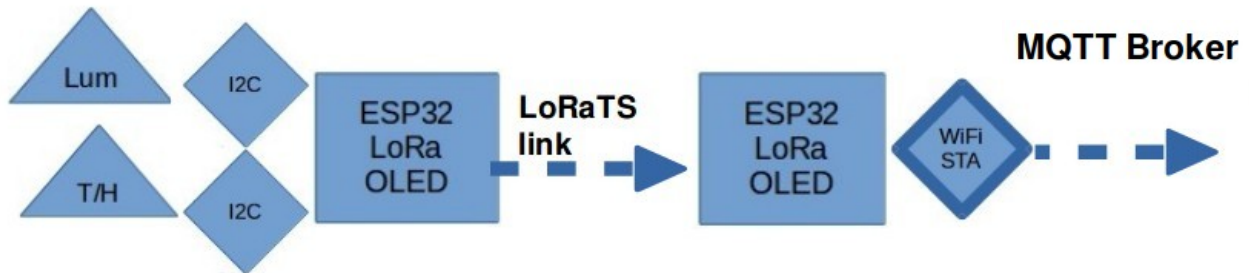
```

## A faire :

1. Tester le code de base
2. Ecrire le code complet pour la version de base avec le transfert du numéro du canal et de clé d'écriture dans les paquets LoRa. Tester ce nouveau programme de passerelle avec un puis avec 2 terminaux.
3. Utiliser les capteurs SHT21 et BH1750 pour envoyer des données « réelles »

## 5.2 Passerelle LoRa – WiFi – broker MQTT

Dans cette partie du laboratoire nous allons développer une passerelle entre un lien LoRa et une connexion WiFi qui porte les messages MQTT vers un broker MQTT.



### 5.2.1 L'émetteur des messages MQTT sur LoRa

```
#include <Arduino.h>

#include <SPI.h>
#include <LoRa.h>
#define SCK 18 // GPIO18 -- SX127x's SCK
#define MISO 19 // GPIO19 -- SX127x's MISO
#define MOSI 23 // GPIO23 -- SX127x's MOSI
#define SS 5 // GPIO05 -- SX127x's CS
#define RST 15 // GPIO15 -- SX127x's RESET
#define DIO 26 // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq 434E6
#define sf 7
#define sb 125E3
union
{
uint8_t frame[64]; // trame avec octets
struct {
char topic[32]; // 4 valeurs en virgule flottante
char mess[32]; // 4 valeurs en virgule flottante
} mqtt;
} sdp ; // paquet d'émission

void setup() {
Serial.begin(9600);
SPI.begin(SCK,MISO,MOSI,SS);
LoRa.setPins(SS,RST,DIO);
Serial.println();delay(100);Serial.println();
if (!LoRa.begin(freq)) { Serial.println("Starting LoRa failed!");while (1);}
Serial.println("Starting LoRa OK!");
LoRa.setSpreadingFactor(sf);
LoRa.setSignalBandwidth(sb);
}

int counter=0;

void loop() // la boucle de l'émetteur
{
Serial.print("New Packet:") ;
LoRa.beginPacket(); // start packet
strcpy(sdp.mqtt.topic, "/esp32/test");
scanf(sdp.mqtt.mess, "%d", counter);
LoRa.write(sdp.frame, 64);
LoRa.endPacket();
Serial.println(counter);
counter++ ;
delay(2000);
}
```

## 5.2.2 La passerelle des messages MQTT sur LoRa vers WiFi et broker MQTT

```
#include <Arduino.h>

#include <SPI.h>
#include <LoRa.h>
#include <WiFi.h>
#include <MQTT.h>

#define NT 4 // max number of terminals
#define NTS 4 // max number of sensors per terminal

const char* ssid = "Livebox-08B0";
const char* pass = "G79ji6dtEptVTPWmZP";

const char* mqttServer = "broker.emqx.io";
//const char* mqttServer = "192.168.1.68";

#define SCK 18 // GPIO18 -- SX127x's SCK
#define MISO 19 // GPIO19 -- SX127x's MISO
#define MOSI 23 // GPIO23 -- SX127x's MOSI
#define SS 5 // GPIO05 -- SX127x's CS
#define RST 15 // GPIO15 -- SX127x's RESET
#define DIO 26 // GPIO26 -- SX127x's IRQ(Interrupt Request)
#define freq 434E6
#define sf 7
#define sb 125E3

union
{
  uint8_t frame[64]; // trame avec octets
  struct {
    char topic[32]; // 4 valeurs en virgule flottante
    char mess[32]; // 4 valeurs en virgule flottante
  } mqtt;
} rdp ; // paquet d'émission

QueueHandle_t msg_queue; // queues for data packets

void onReceive(int packetSize)
{
  int rssi=0;
  uint8_t buff[64];
  if (packetSize == 0) return; // if there's no packet, return
  int i=0;
  if (packetSize==64)
  {
    while (LoRa.available()) { buff[i]=LoRa.read();i++; }
    xQueueReset(msg_queue);
    Serial.println("MQTT packet received");
    xQueueSend(msg_queue, (void *)buff, 64);
  }
}

WiFiClient net;
MQTTClient client;
unsigned long lastMillis = 0;

void connect() {
  Serial.print("checking wifi...");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(1000);
  }
  Serial.print("\nconnecting...");
  while (!client.connect("IoT.GW5")) {
    Serial.print("."); delay(1000);
  }
  Serial.println("\nIoT.GW1 - connected!");
  client.subscribe("/esp32/sensors");
  delay(100);
}
```

```

client.subscribe("/esp32/lorarssi");
}

void messageReceived(String &topic, String &payload) {
Serial.println("incoming: " + topic + " - " + payload);
// send LoRa message depending on topic
}

void setup() {
Serial.begin(9600);
WiFi.begin(ssid, pass);
client.begin(mqttServer, net);
client.onMessage(messageReceived);
connect();
SPI.begin(SCK,MISO,MOSI,SS);
LoRa.setPins(SS,RST,DI0);
Serial.println();delay(100);Serial.println();
if (!LoRa.begin(freq)) {
Serial.println("Starting LoRa failed!");
while (1);
}
msg_queue = xQueueCreate(4,64); // queue for 4 data MQTT packets
Serial.println("Starting LoRa OK!");
LoRa.setSpreadingFactor(sf);
LoRa.setSignalBandwidth(sb);
LoRa.onReceive(onReceive); // pour indiquer la fonction ISR
LoRa.receive(); // pour activer l'interruption (une fois)
}

void loop() {
char vbuff[15];
client.loop();
delay(10); // <- fixes some issues with WiFi stability
if (!client.connected()) { connect(); }
xQueueReceive(msg_queue, (void *)rdp.frame, 6000); // 6s, default:portMAX_DELAY
sprintf(vbuff,"%2.2f",rdp.data[0]);
client.publish(rdp.mqtt.topic, rdp.mqtt.mess);
Serial.printf("published to%s:%s\n",rdp.mqtt.topic,rdp.mqtt.mess);
delay(6000);
}

```

## A faire :

1. Tester le code de base
2. Ecrire le code complet pour la version de base avec le transfert du topic et du message.
3. Utiliser les capteurs SHT21 et BH1750 pour envoyer des données « réelles »

# Table of Contents

SmartComputerLab.....	1
0. Introduction.....	2
0.1 ESP32 Soc – une unité avancée pour les architectures IoT.....	3
0.2 Carte LOLIN D32.....	3
0.3 IoT DevKit une plate-forme de développement IoT.....	4
0.4 L'installation de PlatformIO.....	5
0.4.1 Installation de VSCode.....	5
0.4.2 Installer le package PlatformIO IDE pour VSCode.....	5
0.4.3 Premier démarrage de PlatformIO sur VSCode.....	6
0.4.4 Le menu PIO.....	7
0.4.5 Créer un nouveau projet (ESP32, ESP8266, ... ).....	8
0.4.6 Décryptage du fichier platformio.ini.....	10
0.4.7 Edition du code.....	11
0.7.8 Premier exemple.....	12
Laboratoire 1.....	15
1.1 Premier exemple – l’affichage des données.....	15
A faire:.....	16
1.2 Deuxième exemple – capture et affichage des valeurs.....	17
1.2.1 Capture de la température/humidité par SHT21.....	17
A faire:.....	18
1.2.2 Capture de la luminosité par BH1750.....	19
1.2.3 Capture de la luminosité par MAX44009.....	20
1.2.5 Capture de la pression/température avec capteur BMP180.....	21
1.2.6 Capture de présence avec un capteur PIR SR602.....	22
Laboratoire 2.....	23
Communication en WiFi et broker MQTT.....	23
2.1 Client MQTT – envoi et réception des messages.....	23
A faire :.....	25
2.2 Simple broker MQTT avec ESP8266.....	26
A faire :.....	28
Laboratoire 3 – WiFi avec WiFiManager et serveur ThingSpeak.com (.fr).....	29
3.1 Introduction.....	29
3.1.1 Un programme de test – scrutation du réseau WiFi.....	29
3.2 Mode WiFi – STA, client WEB et serveur ThingSpeak.....	31
A faire.....	32
Attention:.....	33
A faire (comme dans l’exemple précédent) :.....	38
Laboratoire 4 – communication longue distance avec LoRa ( <i>Long Range</i> ).....	39
4.1 Introduction.....	39
4.1.1 Modulation LoRa.....	39
4.1.2 Paquets LoRa.....	39
4.2 Premier exemple – émetteur et récepteur des paquets LoRa.....	41
A faire :.....	43
4.3 onReceive() – récepteur des paquets LoRa avec une interruption.....	44
A faire :.....	45
Laboratoire 5 - Développement de simples passerelles IoT.....	46
5.1 Passerelle LoRa-ThingSpeak.....	46
5.1.1 Le principe de fonctionnement.....	46
5.1.2 Les éléments du code.....	46
5.1.3 Code complet pour une passerelle des paquets en format de base.....	49
A faire :.....	50
5.2 Passerelle LoRa – WiFi – broker MQTT.....	51
5.2.1 L’émetteur des messages MQTT sur LoRa.....	51
5.2.2 La passerelle des messages MQTT sur LoRa vers WiFi et broker MQTT.....	52
A faire :.....	53