Laboratoires IoT avec PlatformIO

SmartComputerLab

Contenu

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 sui 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 oeuvre 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 l'installer les bibliothèque 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 accessibles 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 reçus, 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.

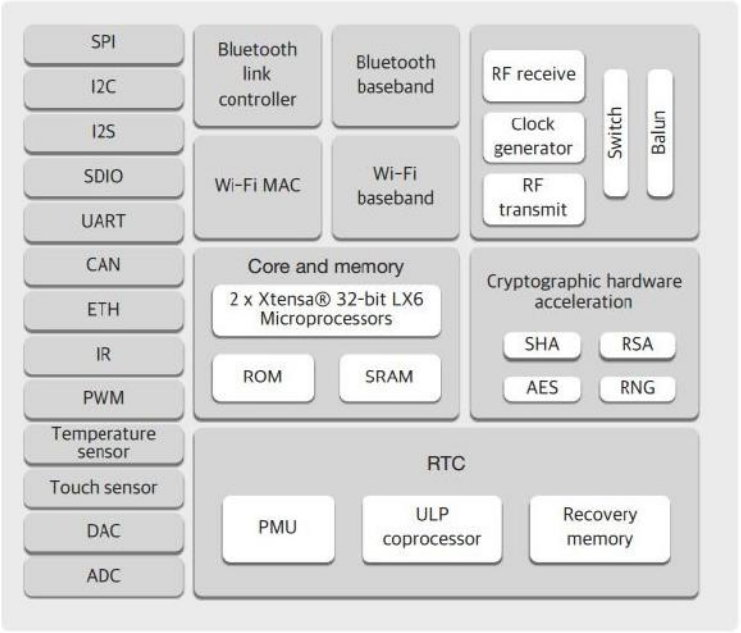


Figure 0.1 ESP32 SoC – architecture interne

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)

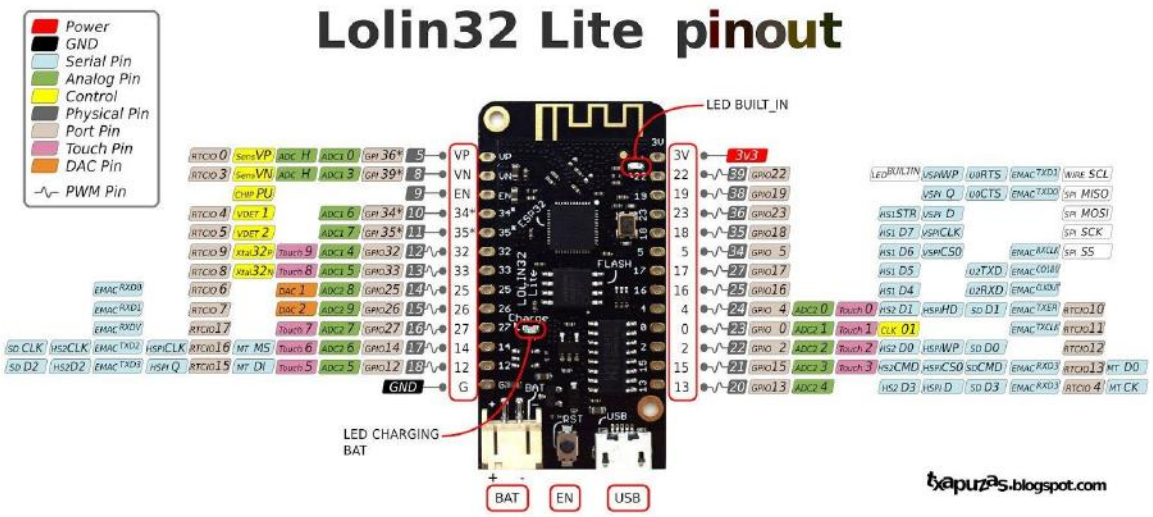


Figure 0.2 Carte MCU **Lolin32 Lite (LOLIN D32) et son *pinout***

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.

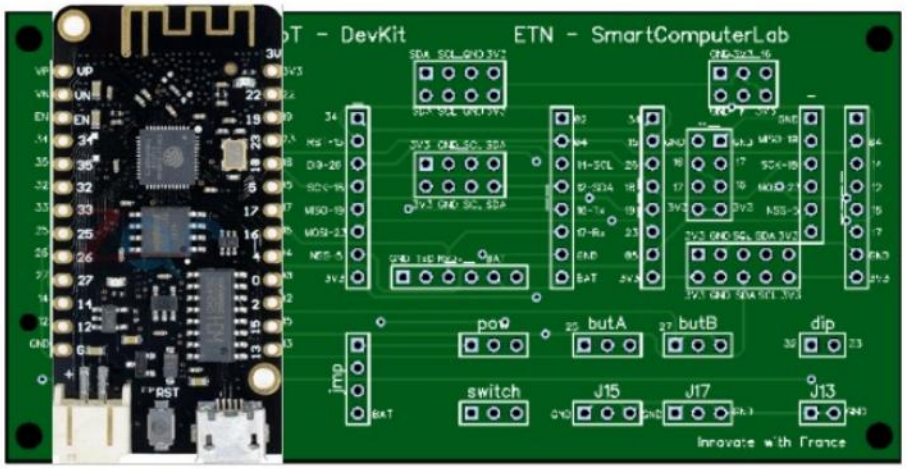


Figure 0.3 Carte de base pour **Lolin32 Lite (LOLIN D32)** et ses interfaces intégrées

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 (**`jmp`**) permet de connecter un multimètre et d'effectuer les mesures du courant.
En mode faible consommation le courant descend à quelques dizaines de micro-ampers.

Ci dessous quelques exemples de cartes d'extension.

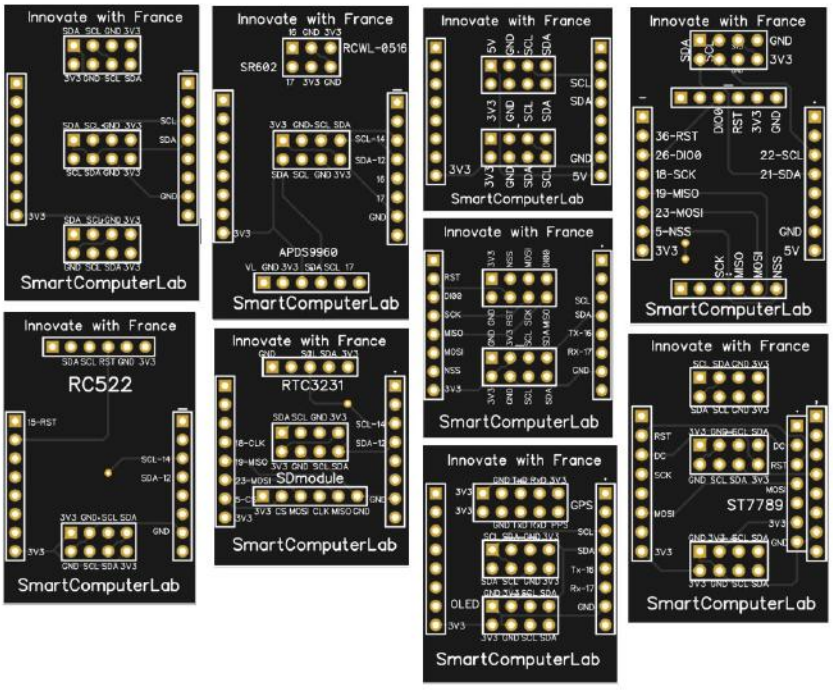


Figure 0.4 Cartes d'extension pour les différents composants IoT: capteurs, afficheurs, modems, ..

0.4 L'installation de `PlatfomIO`

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,

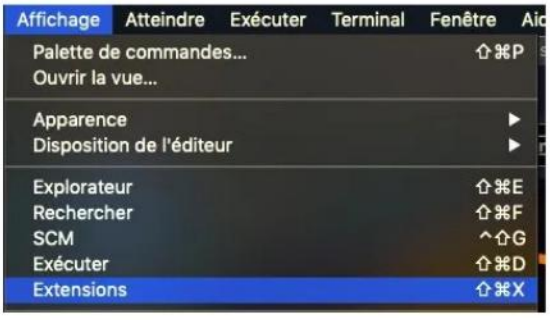
0.4.1 Installation de VSCode

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

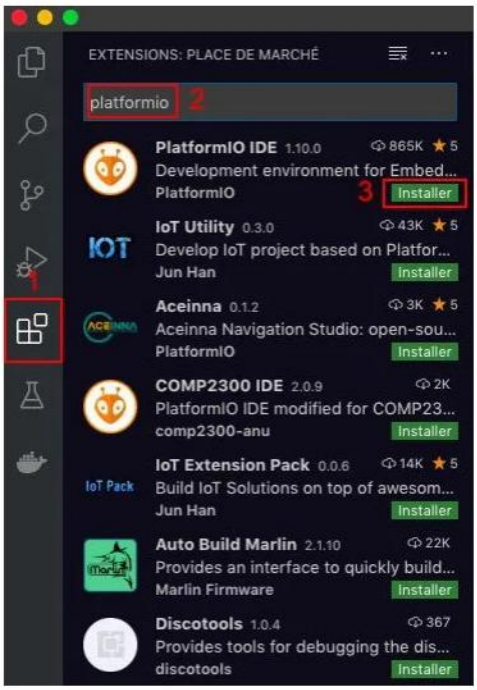
- Windows : installer ou ZIP
- macOS : package dmg
- Linux 32-bits : .deb, .rpm, .tag.gz
- Linux 64-bits : .deb, .rpm, .tag.gz

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.



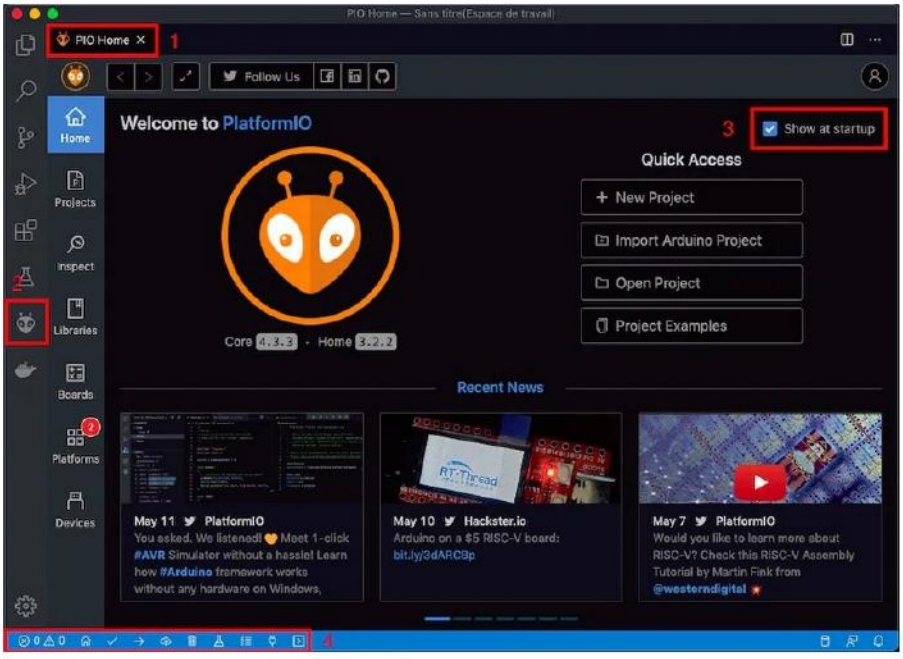
Saisissez **`platformio`** dans le champ de recherche. Cliquez sur **`Install`** pour démarrer l'installation du plugin et des dépendances.



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

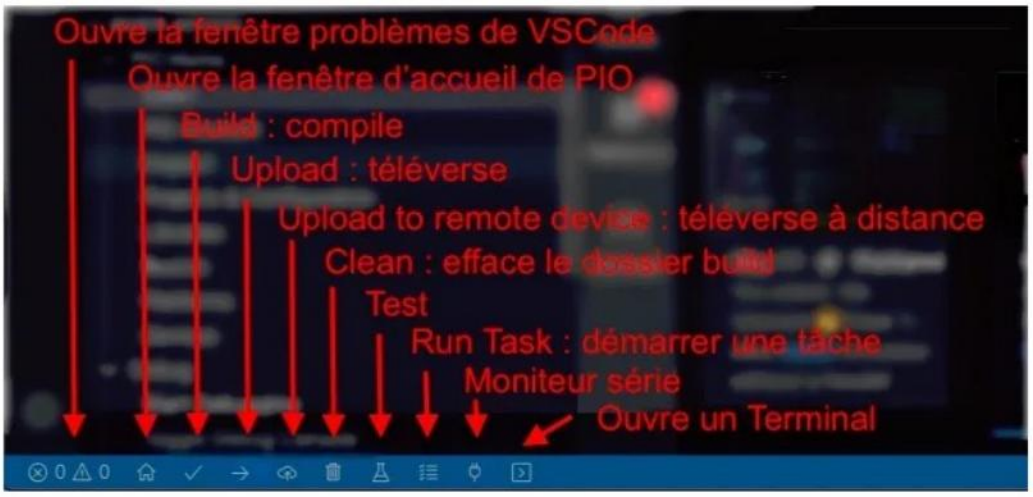
A la fin de l'installation, il n'est pas nécessaire de redémarrer l'éditeur. La page d'accueil de PIO s'ouvre dans un nouvel onglet ①. Cette page ralenti fortement le démarrage de VSCode et n'apporte rien d'utile. Je vous conseille de désactiver son ouverture au démarrage en décochant l'option `Show at startup` ②.
Une nouvelle icône sous la forme d'une tête de fourmi fait son apparition dans la barre latérale ③. Elle permet d'accéder à toutes les fonctionnalités de PIO. Nous allons les détailler un peu plus tard.



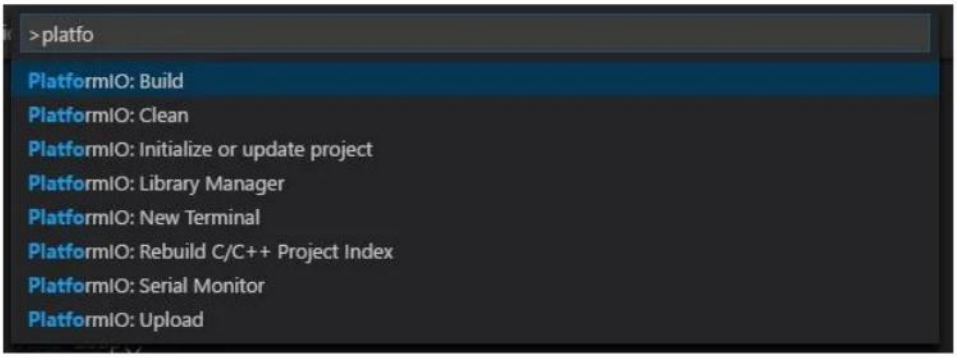
Enfin, une **mini barre** d'outil apparaît dans le bas de l'écran. C'est une version allégée du menu PIO .

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



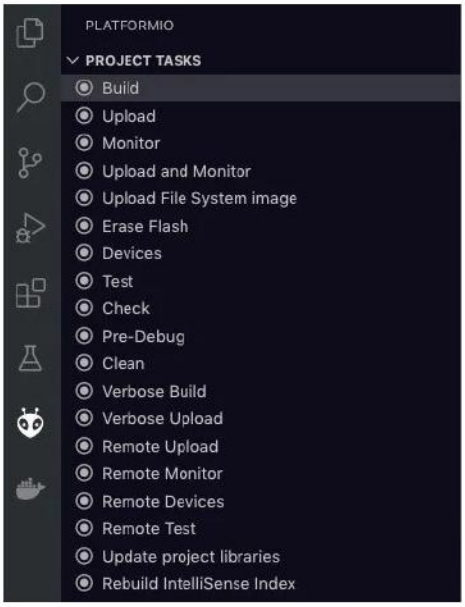
Pour ceux qui préfèrent utiliser les raccourcis clavier, vous pouvez convoquer la palette avec la combinaison de touche **`Ctrl + Shift + P`** (ou **`Cmd + ↑ + P`** sur macOS). Saisissez ensuite le mot clé **`platformio`** pour afficher toutes les commandes disponibles



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

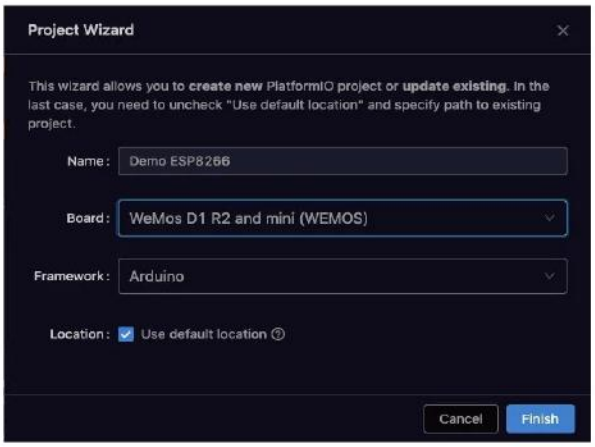
Clean efface le dossier build

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

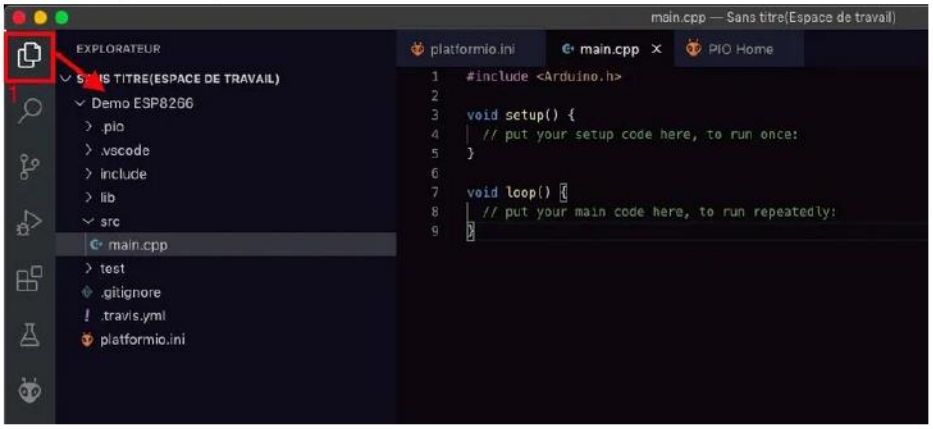
0.4.5 Créer un nouveau projet (ESP32, ESP8266, ...)

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 lettre du fabricant (LOLIN.., Heltec.., WeMos, TTGO..), celle de la carte (D1 mini), le type (**ESP32**, **ESP8266**...) pour filtrer les cartes.

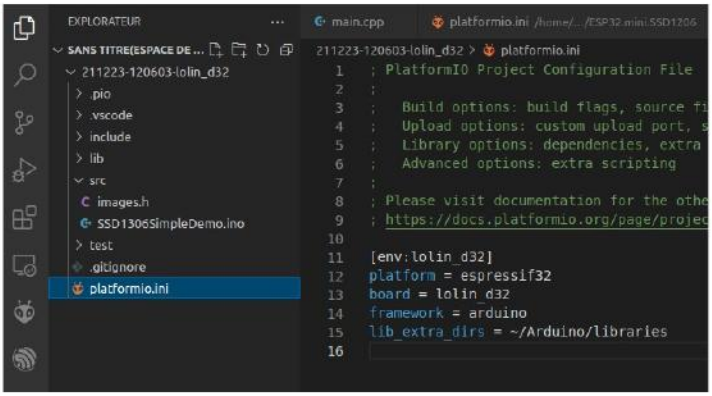


Laissez **`Arduino`** dans le framework. En fonction de votre carte, vous pourrez opter pour un autre framework, mais dans ce cas il faudra vous reporter à la documentation de ce dernier pour la programmation.
Enfin, choisissez le répertoire de création du projet. Si vous cochez **`Use Default Location`**, 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 librairies 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 ce 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 bon 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):

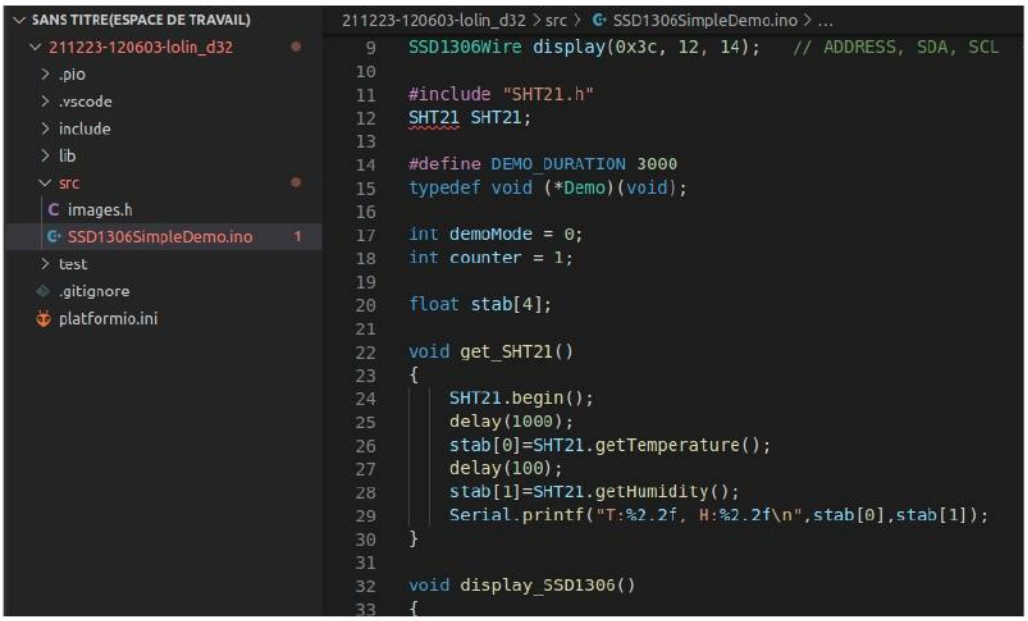
```
[env:heltec_wifi_lora_32_V2]
platform = espressif32
board = heltec_wifi_lora_32_V2
```

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

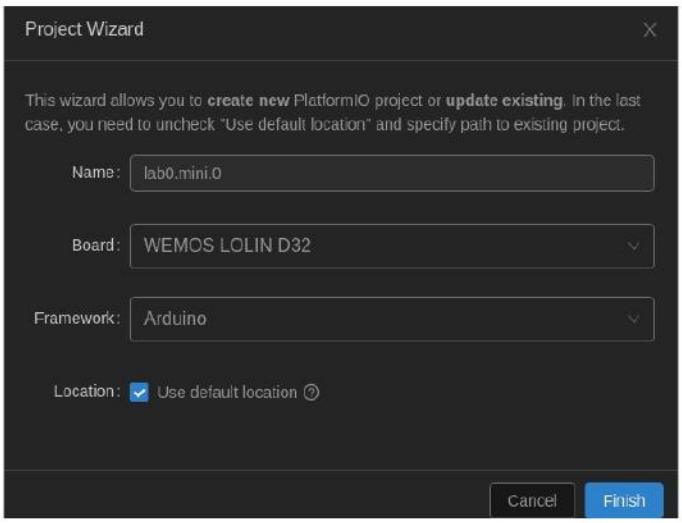
Voici un exemple du code source (projet **`Arduino`** avec extension **`.ino`**) :



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.
```

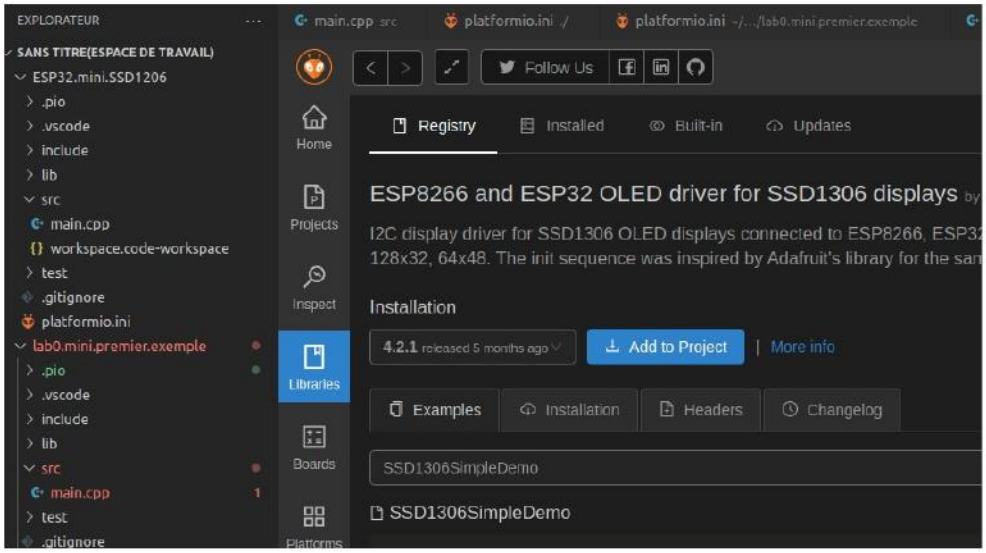
0.7.8 Premier exemple



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

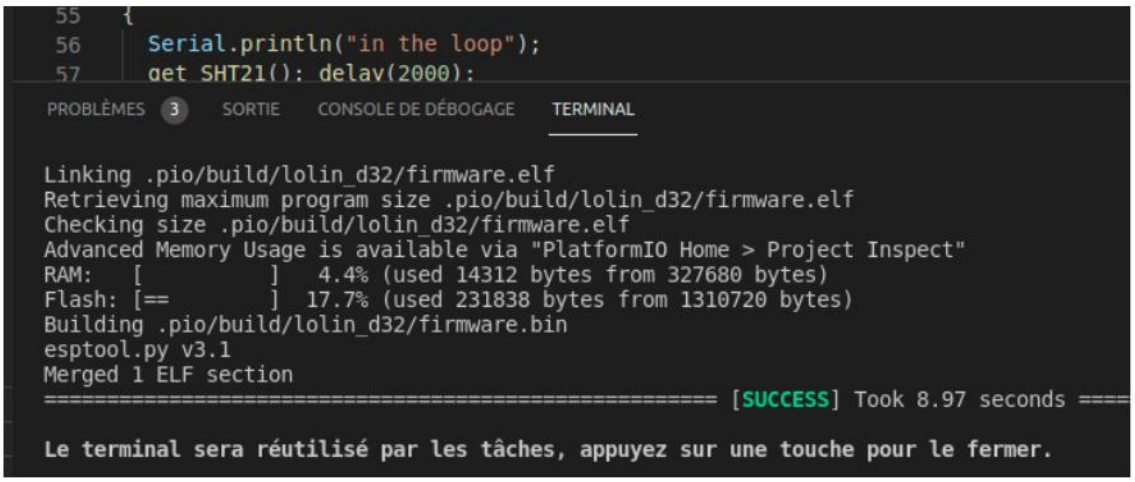
```
; Build options: build flags, source filter
; Upload options: custom upload port, speed and extra flags
; Library options: dependencies, extra library storages
; Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:lolin_d32]
platform = espressif32
board = lolin_d32
framework = arduino
lib_deps =
      https://github.com/markbeee/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);
}
```



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
```