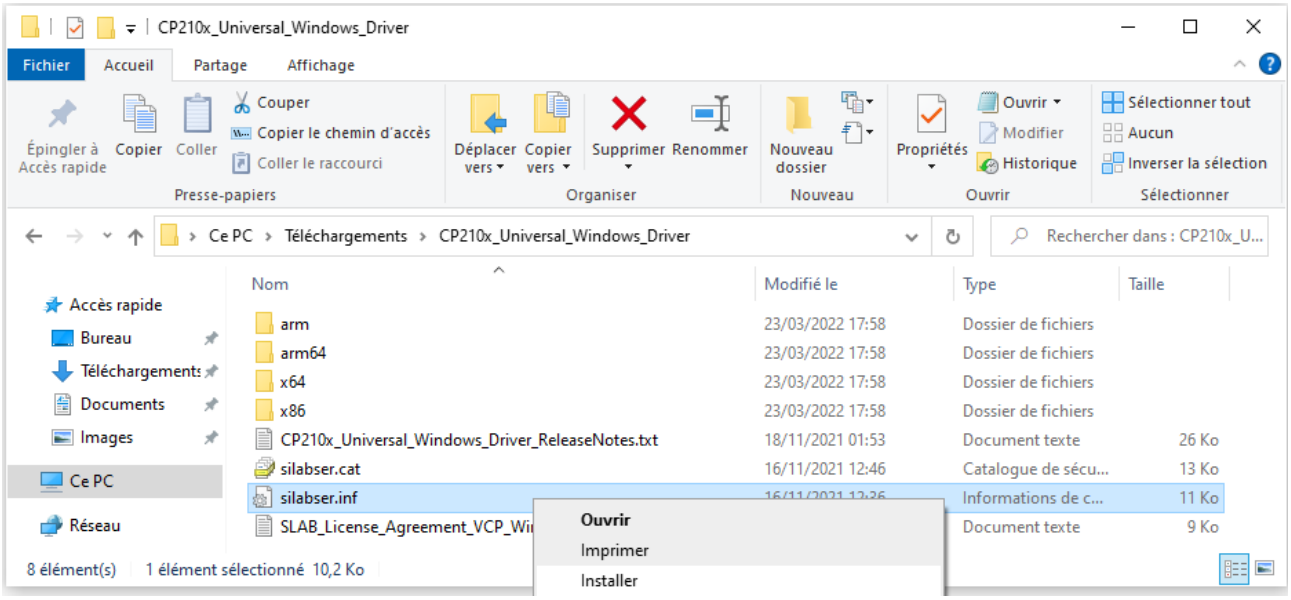
Débuter avec un module Espressif – ESP32, Visual Studio Code et PlatformIO

Installer le driver USB pour les **Espressif – ESP32**
Télécharger le driver USB :
https://www.silabs.com/documents/public/software/CP210x_Universal_Windows_Driver.zip

Dézipper le fichier zip obtenu. Se placer dans le dossier dézippé et faire un clic droit sur le fichier ***silabser.inf*** puis choisir *Installer* et suivre les instructions.

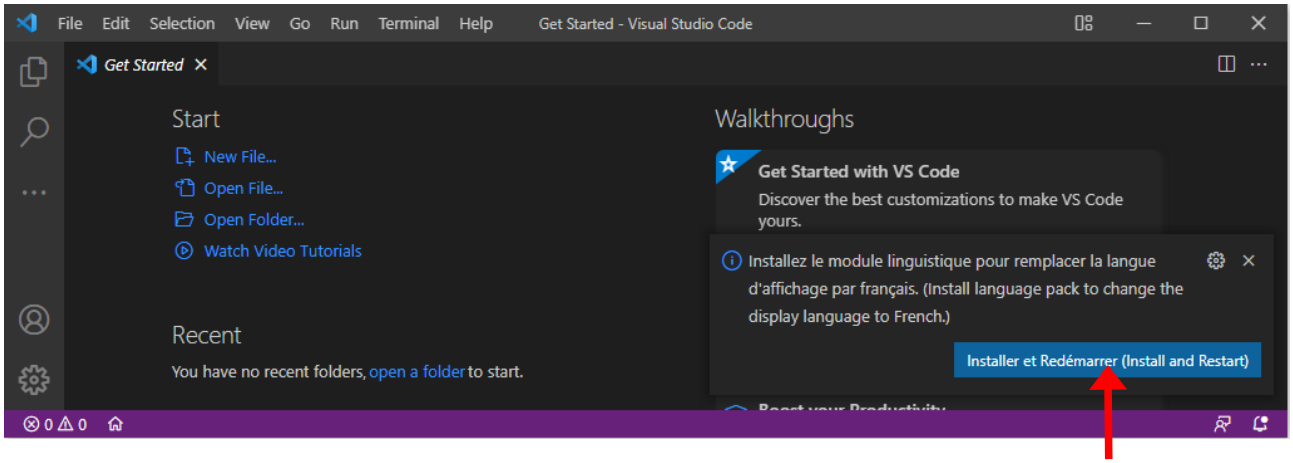


Installation du driver USB

Télécharger **Visual Studio Code** en suivant le lien ci-dessous et effectuer l'installation.
https://code.visualstudio.com/

Exécuter **Visual Studio Code**

Module linguistique (au choix)
Vous pouvez laisser Visual Studio Code en anglais ou passer en français.



Si votre système est en français, Visual Studio Code propose d'installer le module linguistique correspondant

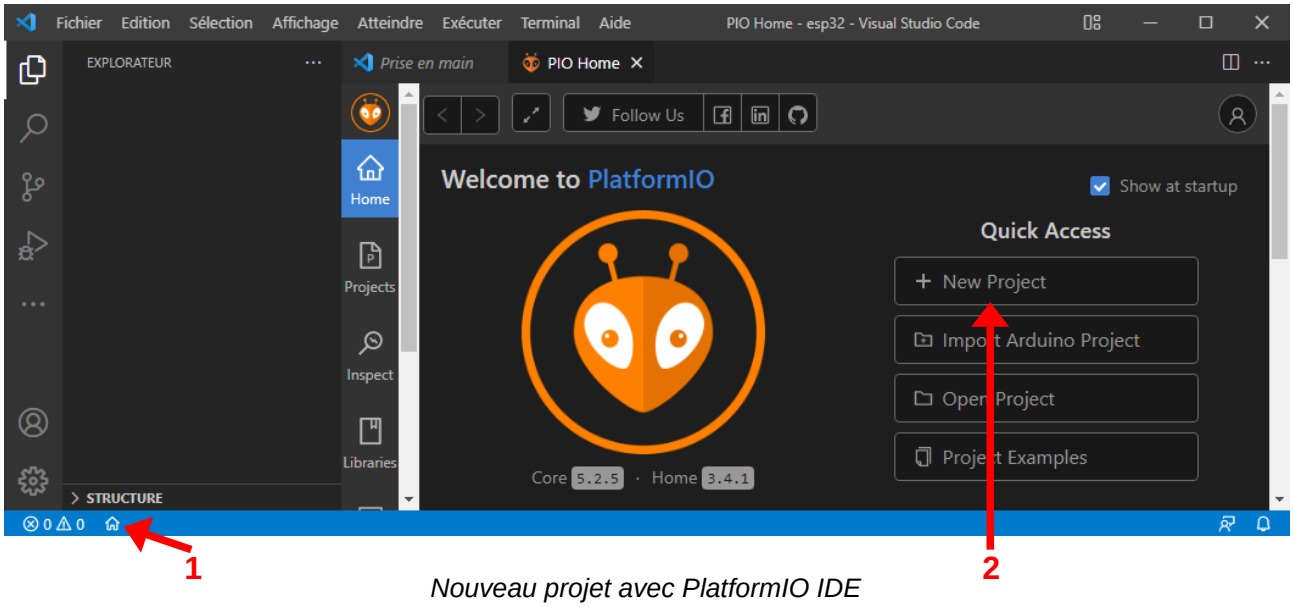
Module linguistique

Installer **PlatformIO IDE**

Installation de PlatformIO IDE

1. Sélectionner *Extensions*
2. Rechercher PlatformIO IDE
3. Sélectionner PlatformIO IDE
4. Installer PlatformIO IDE

Nouveau projet avec PlatformIO IDE

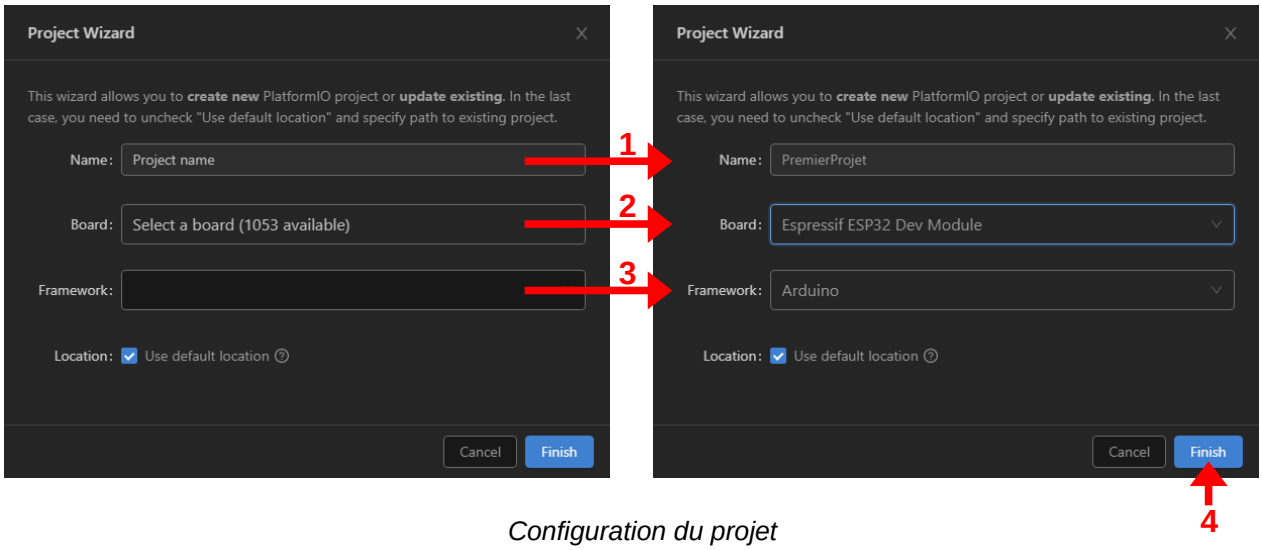


Nouveau projet avec PlatformIO IDE

1. Aller sur l'accueil (*Home*) de PlatformIO IDE en cliquant sur la maison
2. Choisir *New Project*

Le nouveau projet sera créé dans le dossier *~/Documents/PlatformIO/Projects*

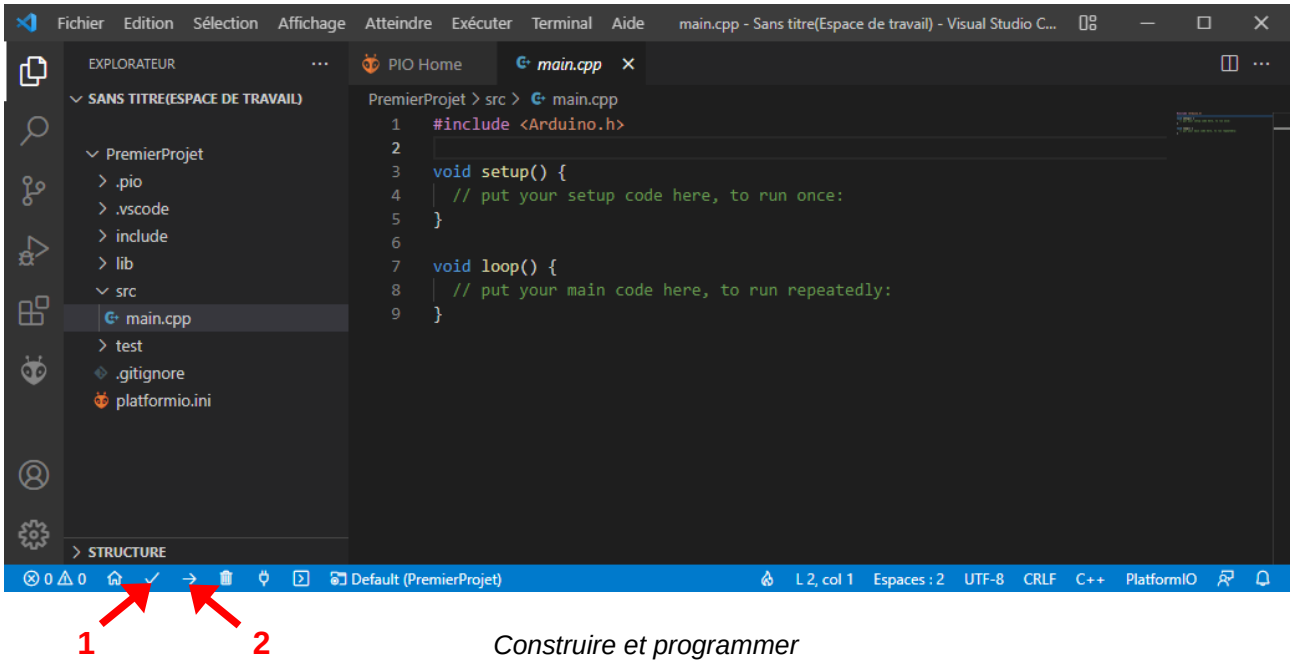
Configurer le nouveau projet



Configuration du projet

1. Choisir un nom de projet
2. Sélectionner la carte (*Board*) utilisée, taper *Espressif* pour trouver plus rapidement dans la liste *Espressif ESP32 Dev Module*
3. Si aucun autre outil n'a été installé sous Visual Studio Code, seul le *Framework* Arduino est disponible
4. Cliquer sur *Finish*

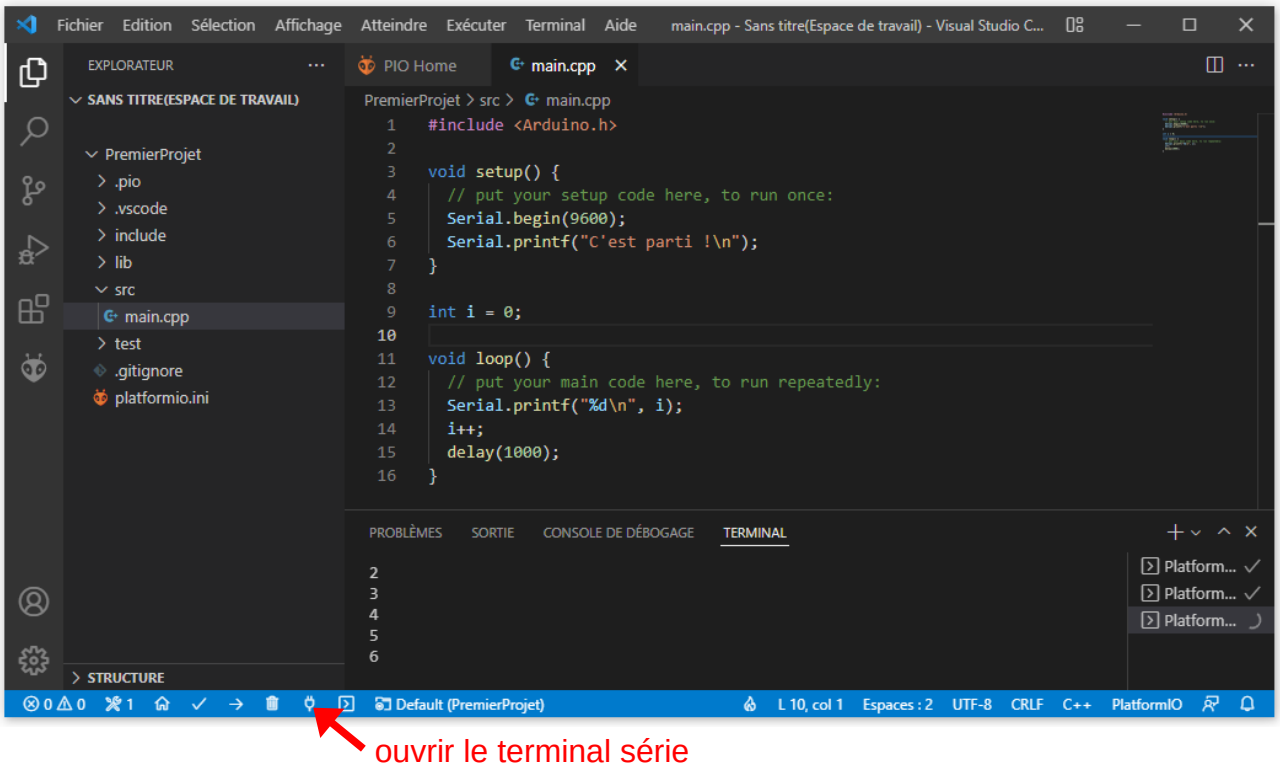
Construire le projet et programmer l'ESP32



Construire et programmer

1. Construire le projet (build)
2. Programmer l'ESP32 (upload)

Premier exemple : utiliser la communication série par le câble USB

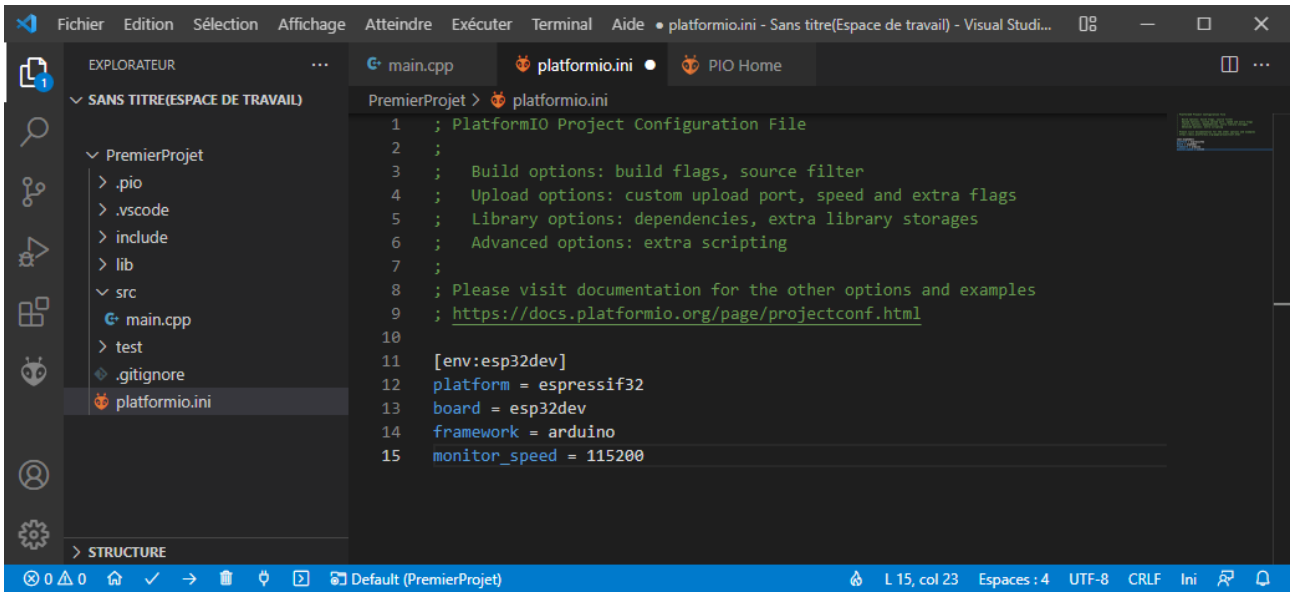


Programme de communication en fonctionnement

Le terminal est configuré par défaut à 9600 bauds. Pour utiliser un autre débit, il faut ajouter un paramètre de configuration dans le fichier *platformio.ini* du projet. Pour un débit de 115200 bauds, on ajoutera la ligne :

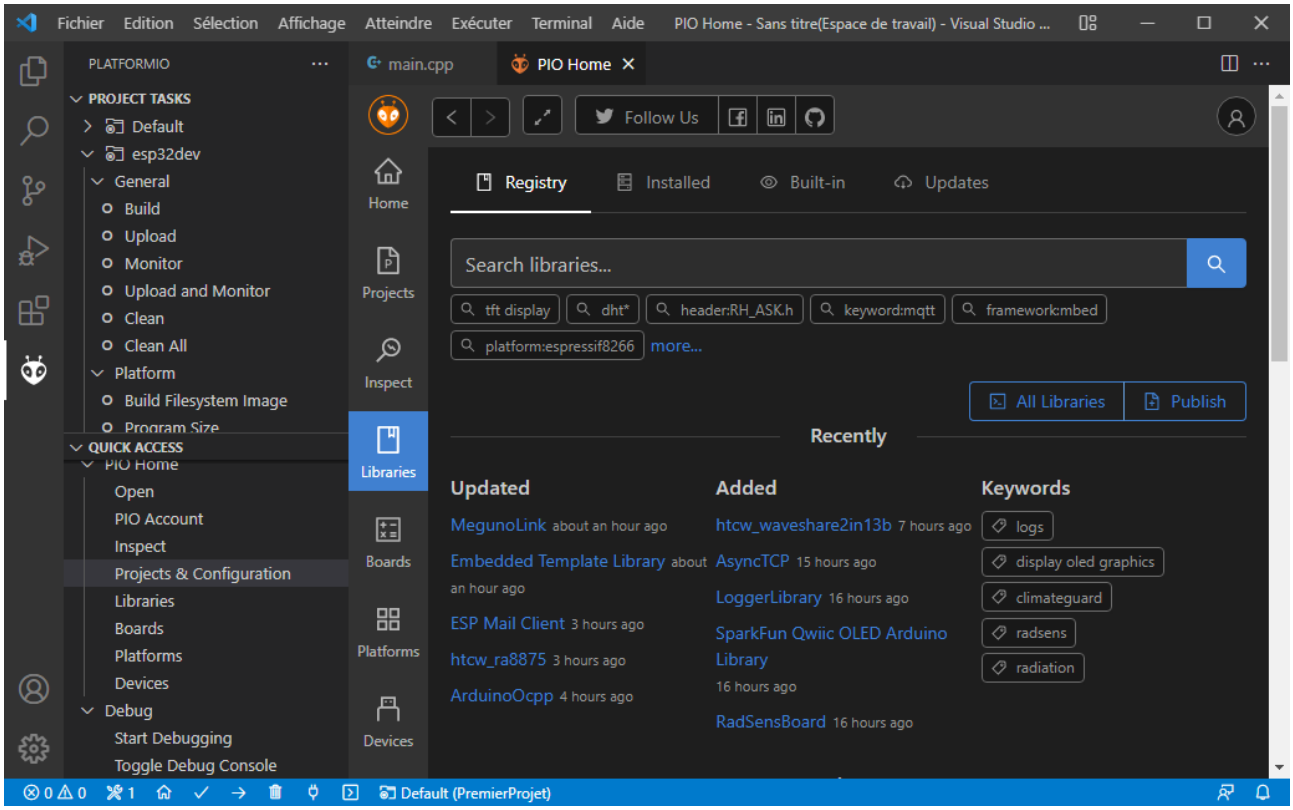
```
monitor_speed = 115200
```

Il faudra modifier en conséquence la valeur du *Serial.begin(**115200**)* dans la fonction *setup* du programme.



Modification du débit de communication du terminal

Accéder aux bibliothèques Arduino



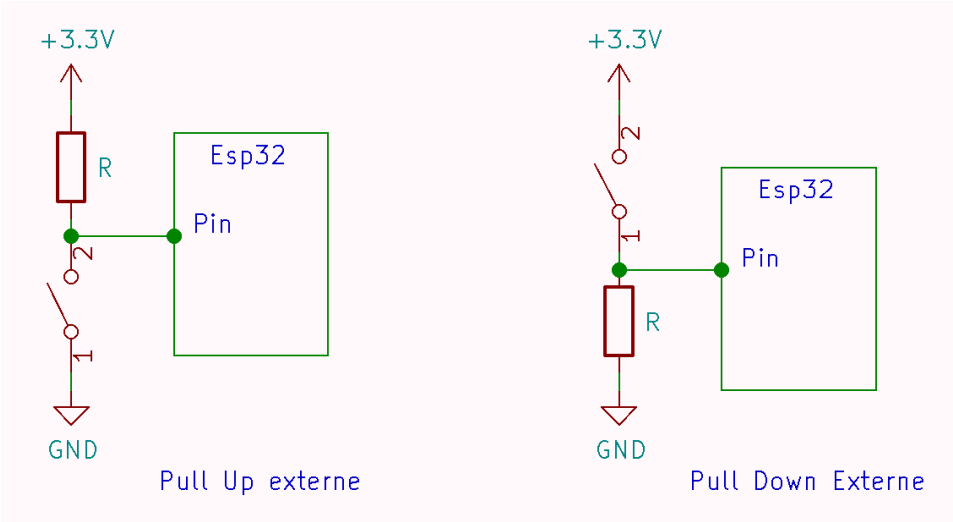
Quatre onglets sont disponibles :

1. Registry : rechercher des bibliothèques pour les installer
2. Installed : lister les bibliothèques déjà installées
3. Built-in : lister les bibliothèques disponibles dans le framework utilisé (ici Arduino pour ESP32)
4. Updates : lister les mises à jour de bibliothèques disponibles

Entrée TOR/ Digital input

Dans cette partie vous allez voir comment configurer une entrée TOR, lire une entrée TOR et afficher le résultat sur le terminal.

Pour rappel il y a 2 possibilités pour connecter un bouton poussoir sur une entrée TOR

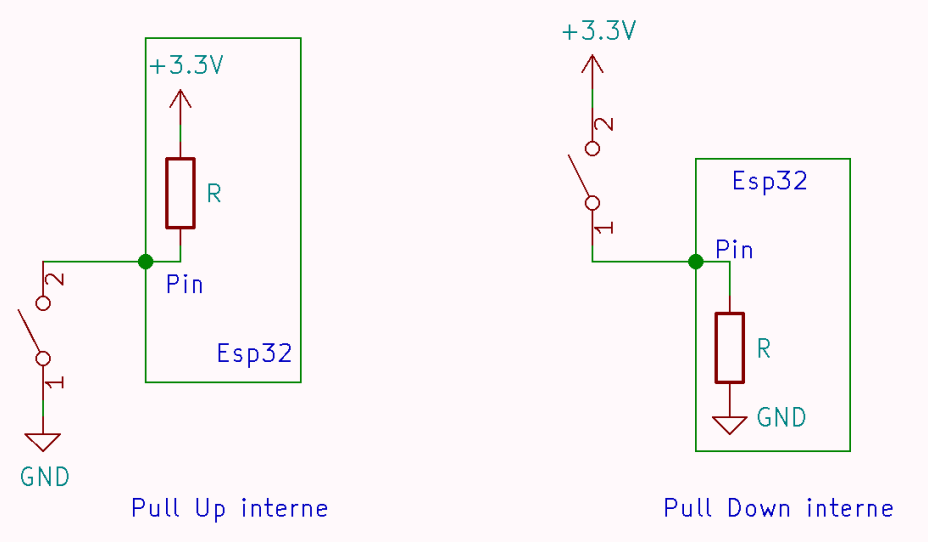


Digital > src > main.cpp > setup()

```
1   #include <Arduino.h>
2
3   // définition des numéros des pins
4   int BP1=32; // BP1 sur la pin 32
5   int BP2=33; // BP1 sur la pin 33
6
7   // déclaration des variables pour lire les entrées
8   int Val_BP1;
9   int Val_BP2;
10
11
12  void setup() {
13    // configuration des entrées
14    pinMode(BP1,INPUT);
15    pinMode(BP2,INPUT);
16
17    // config de la liaison série
18    Serial.begin(9600);
19    }
20
21  void loop() {
22    // lecture des entrées
23    Val_BP1=digitalRead(BP1);
24    Val_BP2=digitalRead(BP2);
25
26   // affichage
27   Serial.printf("bp1 %d  bp2 %d\n",Val_BP1,Val_BP2);
28
29  }
30
```

1. Association des noms des pin avec les numéros des entrées par exemple le bouton poussoir BP1 est connecté à la pin 32 de l'ESP.
2. Déclaration des variables servant à lire les entrés
3. Configuration des pin BP1 et BP2 soit des pin 32 et 33 en entrée TOR
4. Lecture des entrées et stockage dans les variables.
5. affichage des entrées dans le terminal.

Il est possible d'utiliser un pull up ou un pull down interne au microcontrôleur comme le montre la figure ci-dessous. L'intérêt est d'enlever la résistance de pull up ou de pull down dans le design de votre carte électronique.



Pour utiliser les pull up ou les pull down interne il faut modifier la configuration comme ci-dessous.

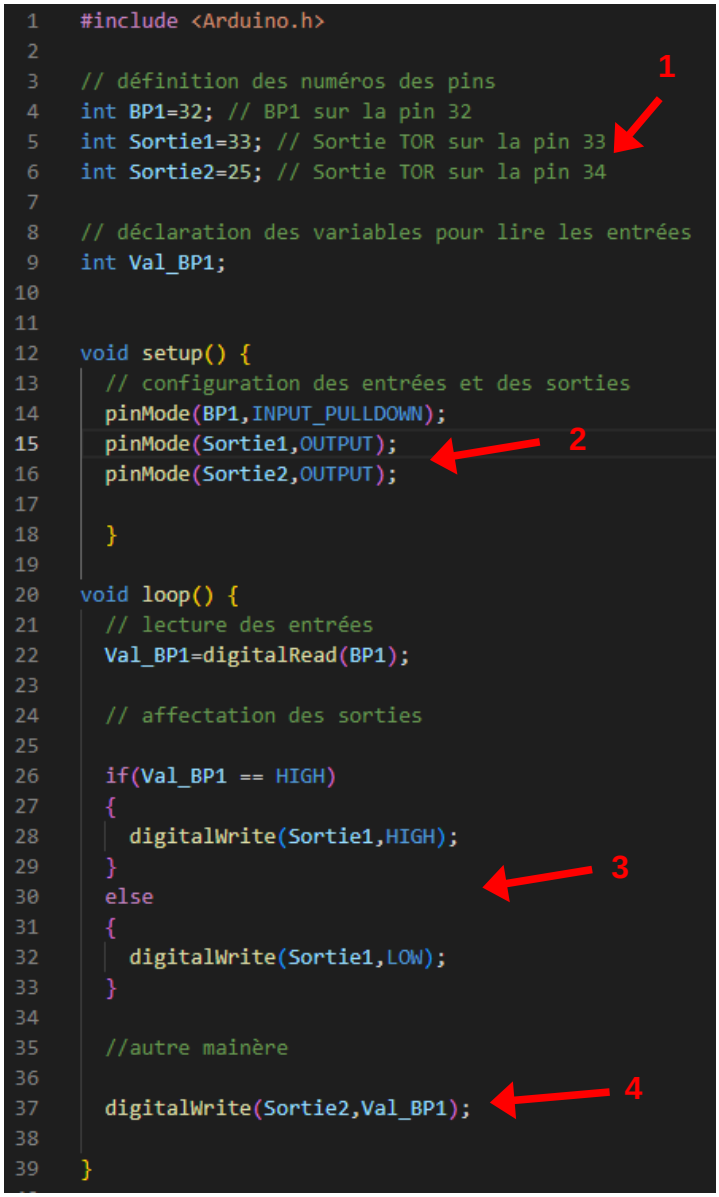
```
void setup() {
  // configuration des entrées
  pinMode(BP1,INPUT_PULLDOWN);
  pinMode(BP2,INPUT_PULLUP);

  // config de la liaison série
  Serial.begin(9600);
  }
```

Dans ce cas BP1 utilise une résistance de pull down interne et BP2 utilise une résistance de pull up interne. Le reste du code reste inchangé.

Sortie TOR/ Digital output

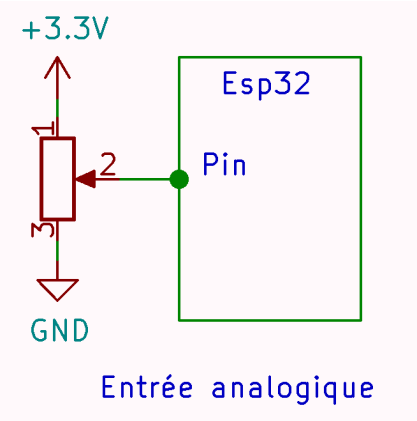
Dans cette partie vous allez voir comment configurer une sortie TOR, donner une valeur à une sortie TOR.



1. Association des noms des pin avec les numéros des sorties par exemple le sortie 1 est affectée à la pin 33 de l'ESP.
2. Configuration des pin Sortie1 et Sortie2 soit des pin 33 et 34 en sorties TOR
3. première méthode pour affecter l'état du bouton poussoir 1 à la sortie1
4. deuxième méthode pour affecter l'état du bouton poussoir 1 à la sortie2.

Entrée analogique

Dans cette partie vous allez voir comment configurer une entrée analogique, lire une entrée analogique et afficher le résultat sur le terminal.
L'ESP32 utilise un convertisseur 12 bits soit une valeur comprise entre 0 et 4095. Pour utiliser une entrée analogique il est possible de brancher un potentiomètre comme ci-dessous

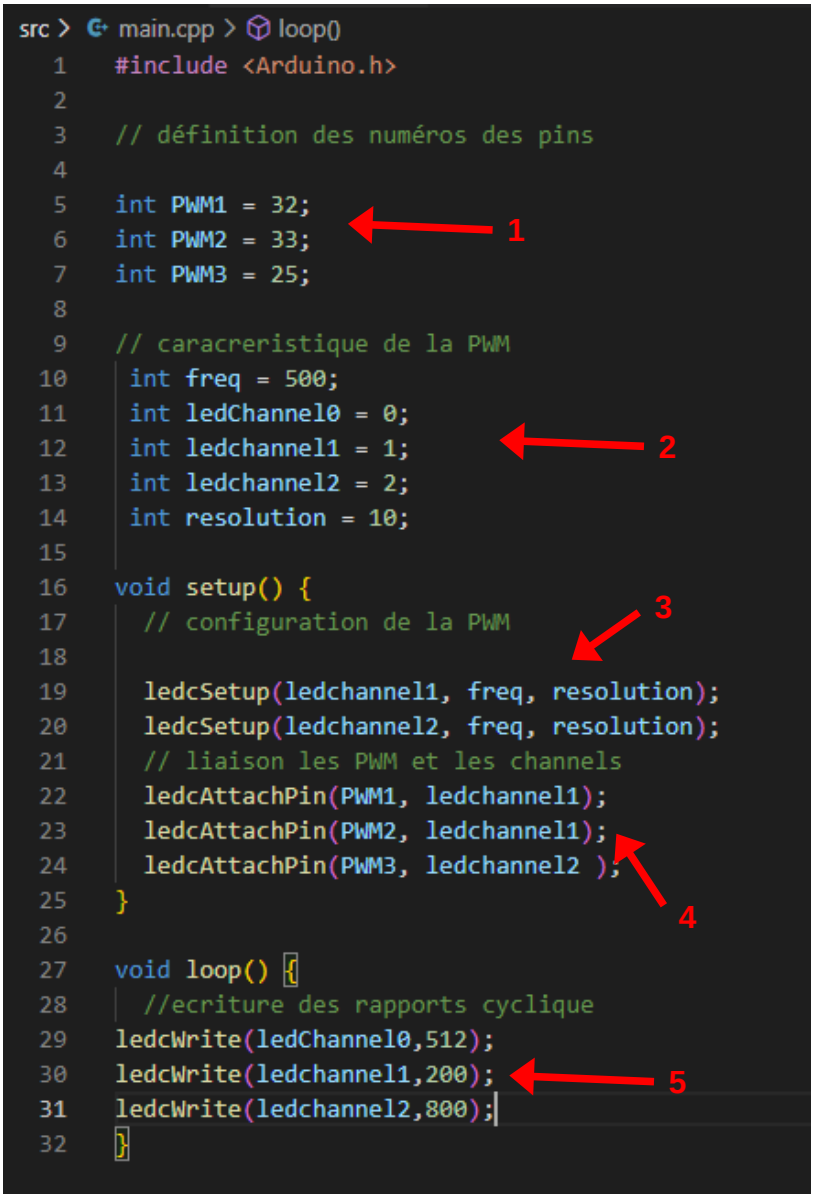



```
1  #include <Arduino.h>
2
3  // définition des numéros de pins
4  int pot = 33; // potentiometre sur la pin 33
5
6  void setup() {
7    // configuration de la liaison série
8    Serial.begin(9600);
9    Serial.printf("coucou\n");
10
11 }
12 //declaration des variables pour lire les entrées
13 int lecture_pot;
14
15 void loop() {
16
17   //Lecture de l'entrée analogique
18   lecture_pot=analogRead(pot);
19   // affichage
20   Serial.printf("pot=%d\n",lecture_pot);
21
22 }
```

1. Association du nom de la pin avec le numéro de l'entrée, le potentiomètre est connecté à la pin 33 de l'ESP.
2. Déclaration de la variable servant à lire l'entrée
3. Lecture de l'entrée et stockage dans la variable lecture_pot.
4. affichage dans le terminal.

Sortie PWM

L'ESP32 dispose de 16 Channels, de 0 à 15, pour les PWM. Il est possible de configurer la fréquence et la résolution des channles (de 1 à 16 bits). Mais attentions les channels sont associés par 2 c'est à dire que channel0 et channel1 auront obligatoirement la même fréquence et même résolution. Il est recommandé d'utiliser une résolution 10 bits ce qui donne une valeur de rapport cyclique comprise entre 0 et 1023.



1. Association du nom de la pin avec le numéro de l'entrée, par exemple la sortie PWM1 est associée à la pin 32
2. Définition des fréquences et des résolutions
3. Configuration des PWM (fréquence et résolution) pour rappel pas besoin de définir channel0 puisqu'il est configuré par channel1
4. Association des PWM avec les channels
4. Ecriture des rapports cyclique.