

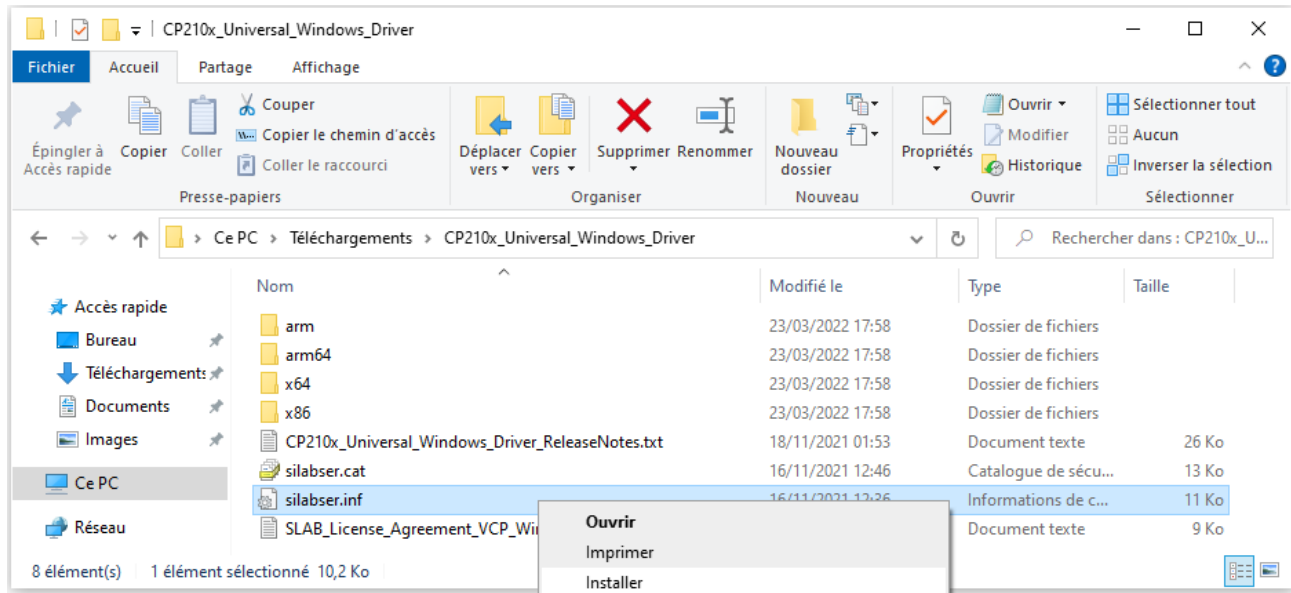
Débuter avec un module Espressif – ESP32, Visual Studio Code et PlatformIO

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

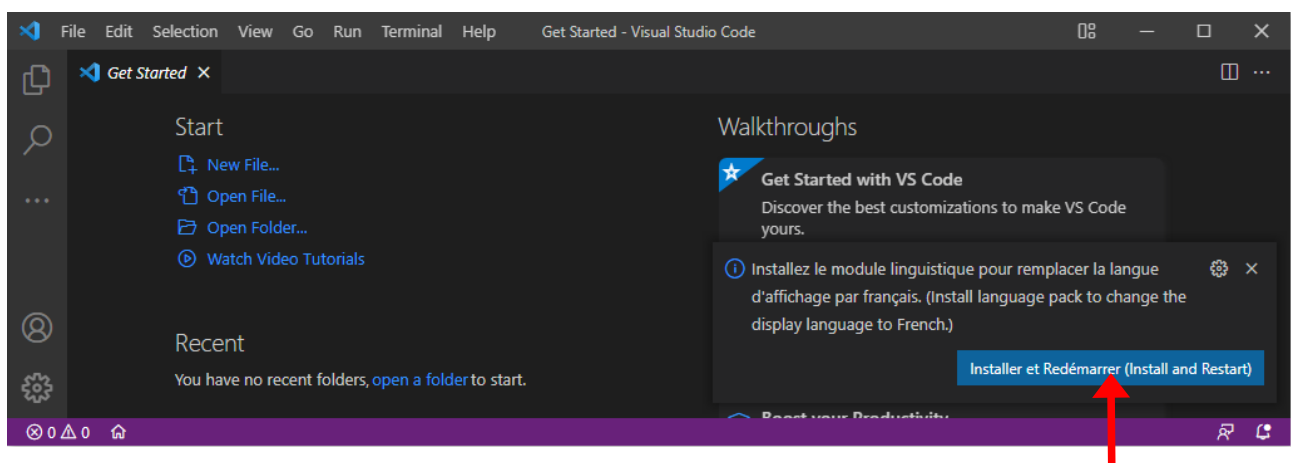
Télécharger **Visual Studio Code** en suivant le lien ci-dessous et effectuer l'installation.

<https://code.visualstudio.com/>

Exécuter **Visual Studio Code**

Module linguistique (au choix)

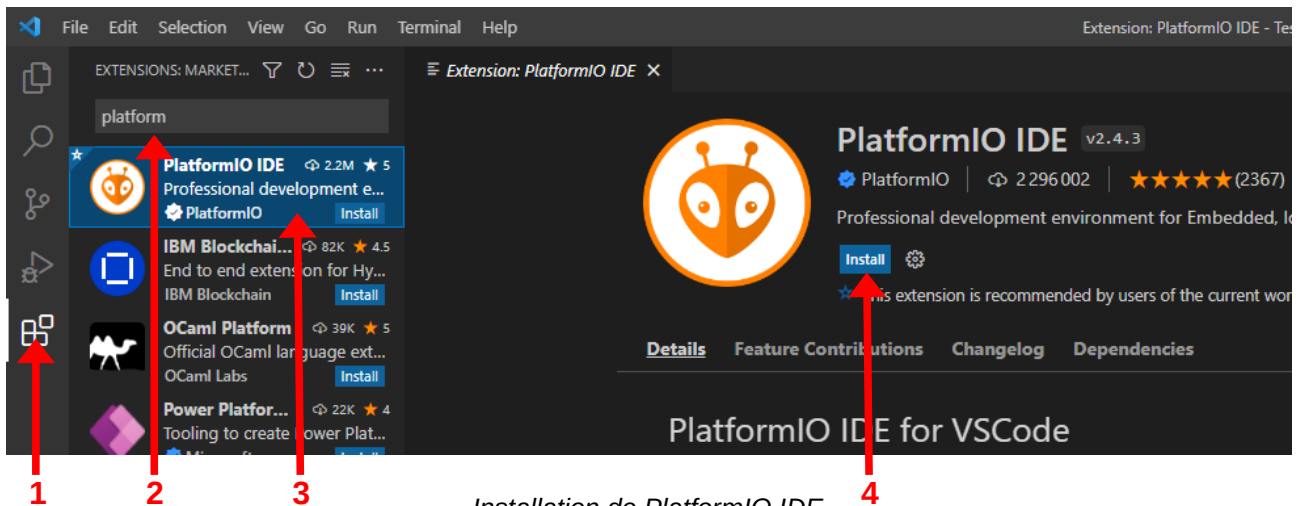
Vous pouvez laisser Visual Studio Code en anglais ou passer en français.



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

Module linguistique

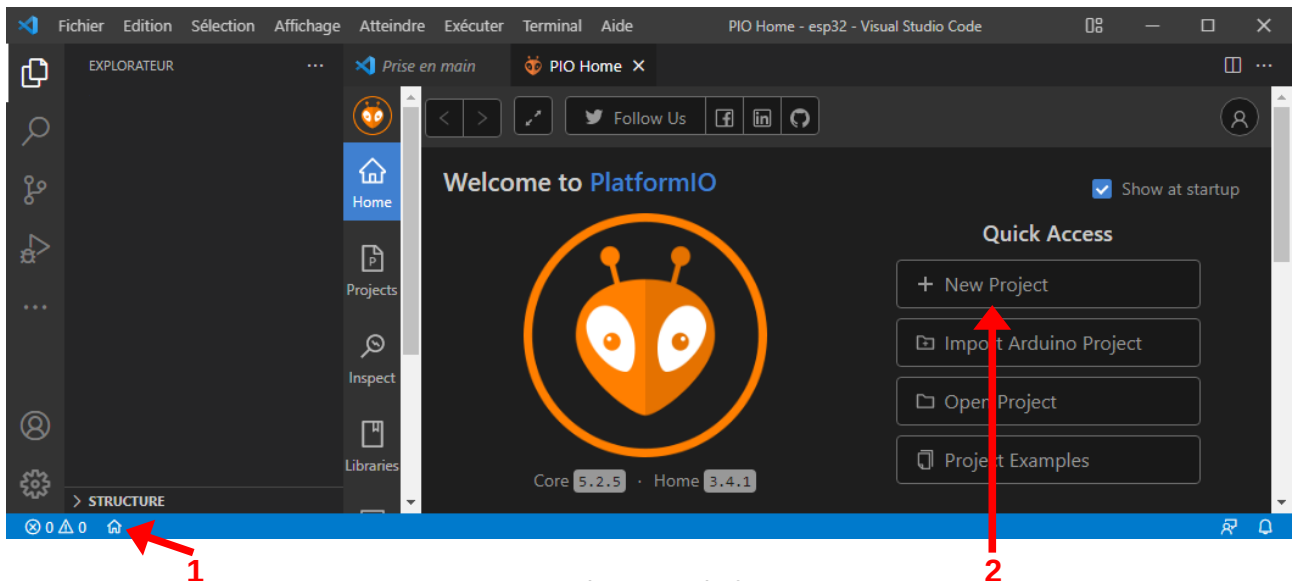
Installer PlatformIO IDE



Installation de PlatformIO IDE

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

Nouveau projet avec PlatformIO IDE

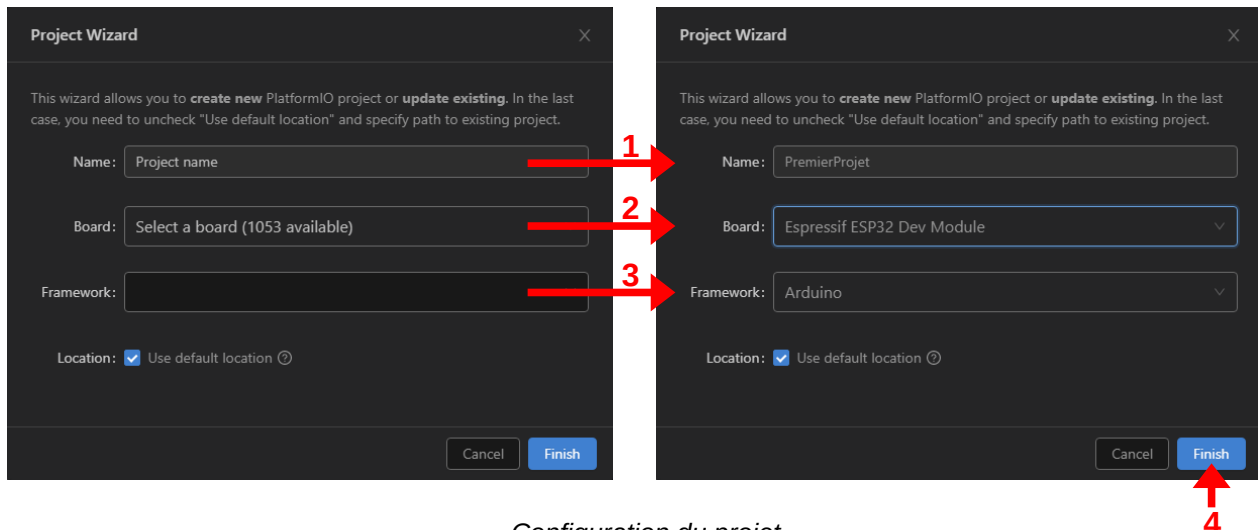


Nouveau projet avec PlatformIO IDE

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

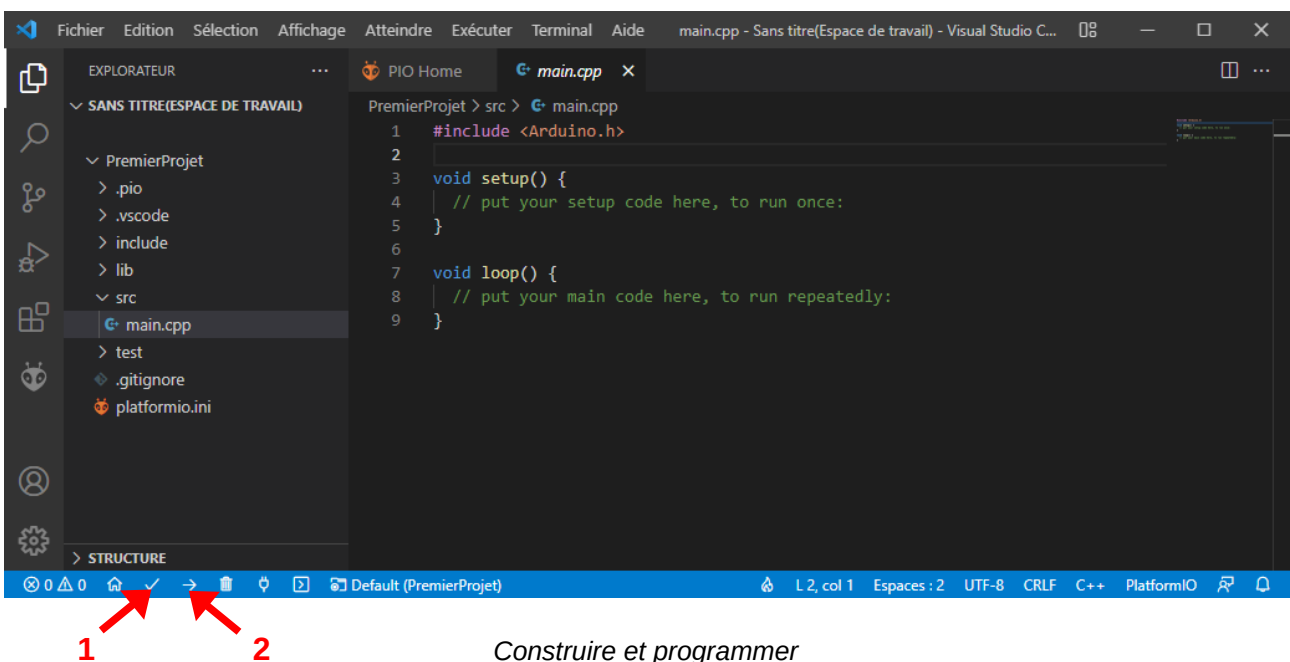
Le nouveau projet sera créé dans le dossier `~/Documents/PlatformIO/Projects`

Configurer le nouveau projet



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



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

La fonction `set up` contient le code de configuration.

La fonction `Loop` contient le code à répéter.

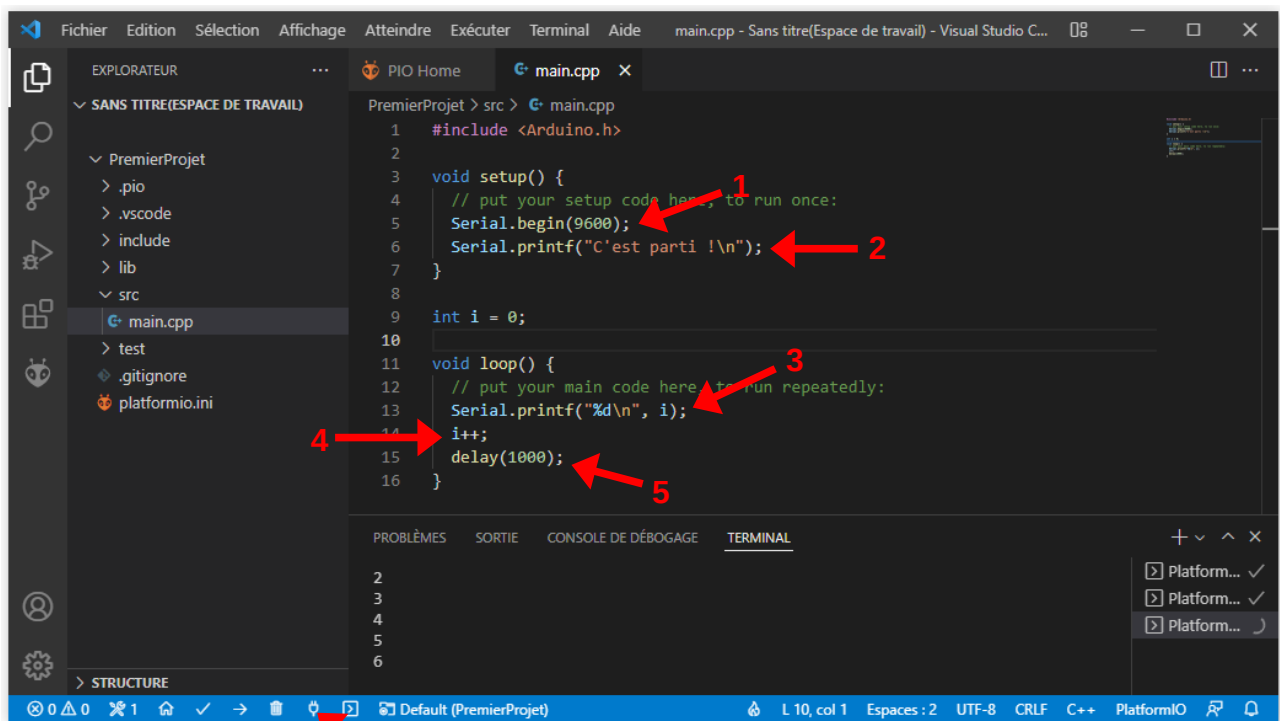
Par rapport à la programmation sur microcontrôleur du semestre 1, c'est un peu comme si on avait le code suivant :

```
void main()
{
    setup();

    while (1)
    {
        loop();
    }
}
```

Utiliser la communication série par le câble USB

Nous allons utiliser le câble USB de programmation pour communiquer entre l'ordinateur et le microcontrôleur. Les bits d'information sont envoyés les uns après les autres, on parle de liaison série. Le débit de transmission s'exprime en bauds.



ouvrir le terminal série

Programme de communication en fonctionnement

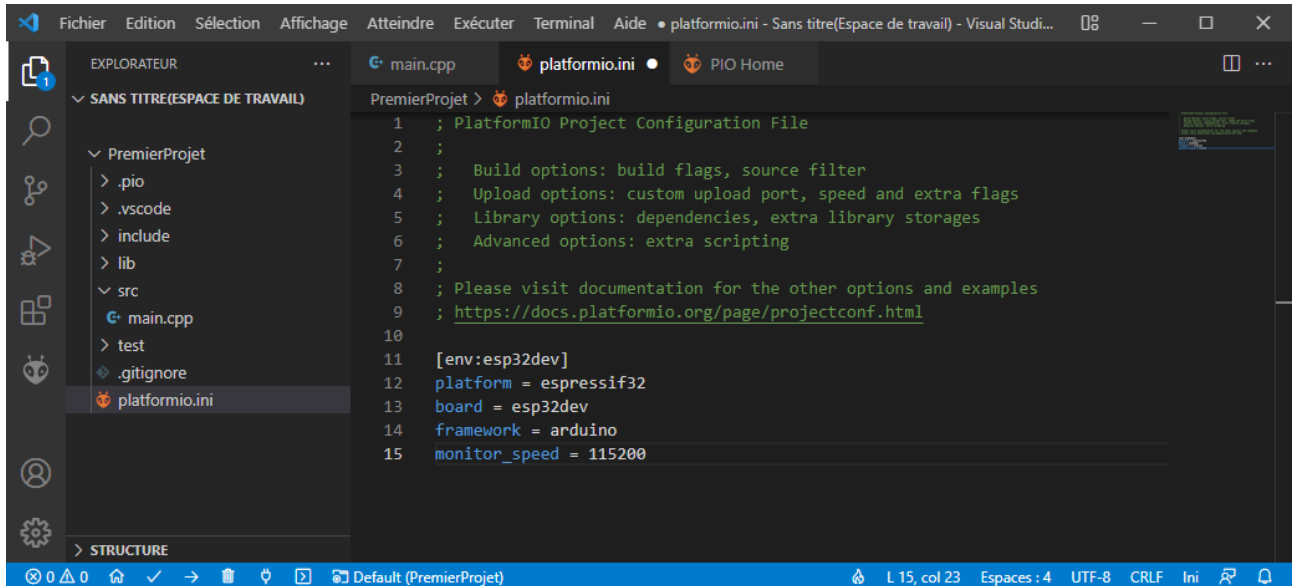
Ce programme affiche la valeur de la variable `i` toutes les secondes grâce à la fonction `delay` qui attend ici 1000 ms.

1. Configurer le débit de la liaison série à 9600 bauds.
2. Afficher un message.
3. Afficher la valeur de la variable `i`.
4. Incrémenter la variable `i`.
5. Attente de 1000 ms.

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.

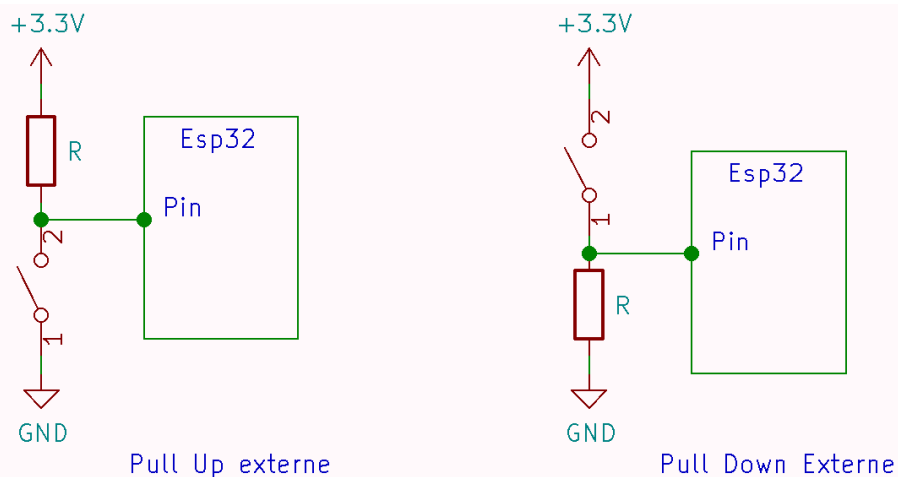


Modification du débit de communication du terminal

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.



Connexion d'un bouton sur une entrée du microcontrôleur avec résistance externe

```

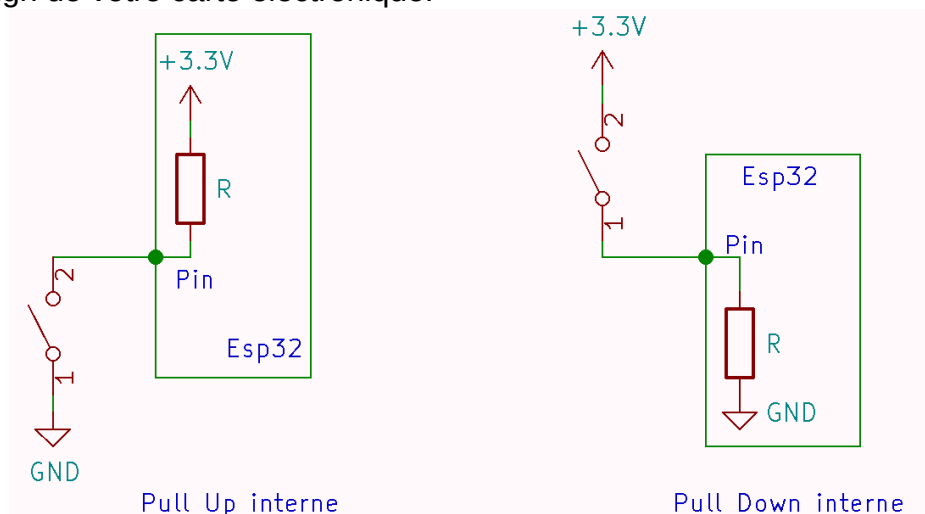
1  #include <Arduino.h>
2
3  // définition des numéros des pins
4  int BP1=32; // BP1 sur la pin 32
5  int BP2=33; // BP1 sur la pin 33
6
7  // déclaration des variables pour lire les entrées
8  int Val_BP1;
9  int Val_BP2;
10
11
12 void setup() {
13     // configuration des entrées
14     pinMode(BP1,INPUT);
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 }

```

Exemple d'utilisation d'entrées TOR

1. Association des noms des broches (pin) avec les numéros des entrées. Sur cet exemple, le bouton poussoir BP1 est connecté à la broche 32 de l'ESP.
2. Déclaration des variables servant à lire les entrées.
3. Configuration des broches pour BP1 et BP2 sur 32 et 33 en entrées 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.



Connexion d'un bouton sur une entrée du microcontrôleur avec résistance interne

Pour utiliser les pull-up ou les pull-down internes 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);  
}
```

Configuration d'un pull-up ou d'un pull-down internes

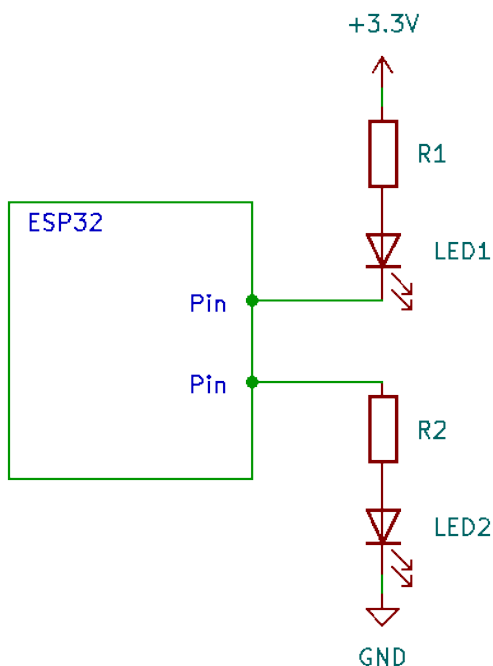
Dans ce cas BP1 utilise une résistance de pull-down interne et BP2 utilise une résistance de pull-up interne. Le reste du code est inchangé.

Sortie TOR / Digital Output

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

Avec les connexions ci-dessous :

- la LED1 s'éclaire avec un état bas, c'est-à-dire 0 V sur la broche de sortie.
- la LED2 s'éclaire avec un état haut, c'est-à-dire 3,3 V sur la broche de sortie.



```
1  #include <Arduino.h>  
2  
3  // définition des numéros des pins  
4  int BP1=32; // BP1 sur la pin 32  
5  int Sortie1=33; // Sortie TOR sur la pin 33  
6  int Sortie2=25; // Sortie TOR sur la pin 34  
7  
8  // déclaration des variables pour lire les entrées  
9  int Val_BP1;  
10  
11  
12 void setup() {  
13   // configuration des entrées et des sorties  
14   pinMode(BP1,INPUT_PULLDOWN);  
15   pinMode(Sortie1,OUTPUT);  
16   pinMode(Sortie2,OUTPUT);  
17  
18 }  
19  
20 void loop() {  
21   // lecture des entrées  
22   Val_BP1=digitalRead(BP1);  
23  
24   // affectation des sorties  
25  
26   if(Val_BP1 == HIGH)  
27   {  
28     digitalWrite(Sortie1,HIGH);  
29   }  
30   else  
31   {  
32     digitalWrite(Sortie1,LOW);  
33   }  
34  
35   //autre manière  
36  
37   digitalWrite(Sortie2,Val_BP1);  
38  
39 }
```

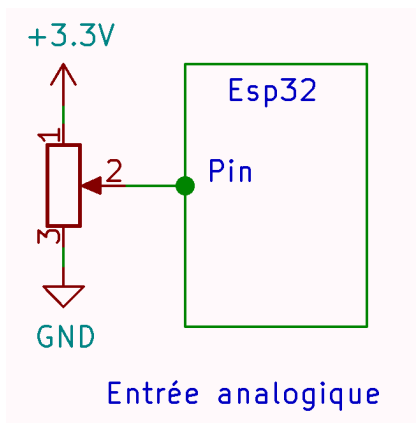
Connexions de leds et programme d'écriture de sorties TOR

1. Association des noms des broches avec les numéros des sorties par exemple la sortie 1 est affectée à la broche 33 de l'ESP.
2. Configuration des broches Sortie1 et Sortie2 soit des broches 33 et 25 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 / Analog Input

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 (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 //déclaration 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 }
```

Connexion d'un potentiomètre et programme de lecture d'une entrée analogique

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

Sortie PWM

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


```

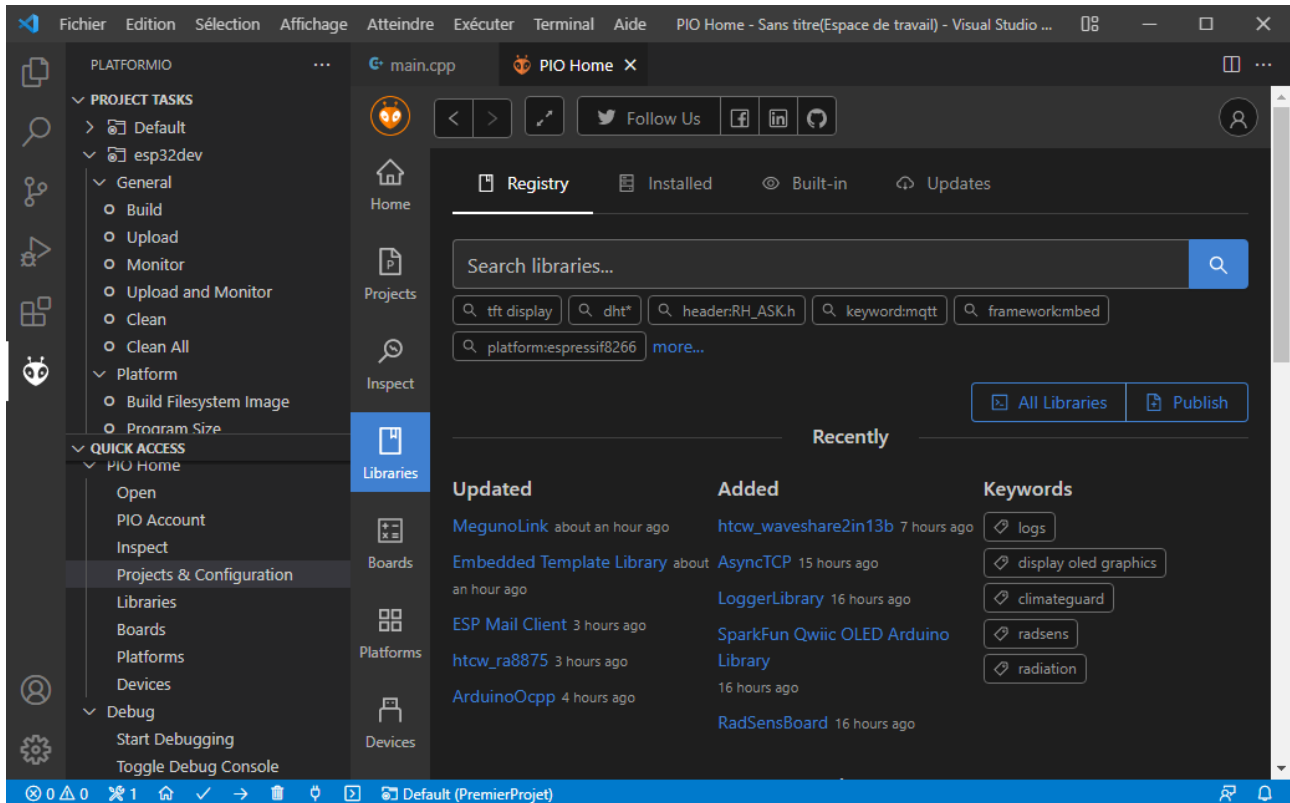
1  #include <Arduino.h>
2
3  // définition des numéros des broches
4  int pwmA = 32;
5  int pwmB = 33;
6  int pwmC = 25;
7
8  // caractéristiques de la PWM
9  int frequence = 500;
10 int canal0 = 0;
11 int canal1 = 1;
12 int canal2 = 2;
13 int resolution = 10;
14
15 void setup() {
16     // configuration de la PWM
17
18     ledcSetup(canal1, frequence, resolution);
19     ledcSetup(canal2, frequence, resolution);
20     // liaison des canaux des PWM avec les broches de l'ESP32
21     ledcAttachPin(pwmA, canal0);
22     ledcAttachPin(pwmB, canal1);
23     ledcAttachPin(pwmC, canal2);
24 }
25
26 void loop() {
27     // put your main code here, to run repeatedly:
28
29     // exemple d'écriture des rapports cycliques
30     // entre 0 et 1023 car résolution 10 bits
31     ledcWrite(canal0, 512);
32     ledcWrite(canal1, 200);
33     ledcWrite(canal2, 800);
34 }

```

Exemple d'utilisation de sorties PWM

1. Association du nom de la broche avec le numéro de l'entrée, par exemple la sortie pwmA est associée à la broche 32.
2. Définition des fréquences et des résolutions.
3. Configuration des PWM (fréquence et résolution), pour rappel pas besoin de définir canal0 puisqu'il est configuré par canal1.
4. Association des PWM avec les *channels*.
5. Écriture des rapports cycliques.

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