



Atelier Arduino Confirmés

Sommaire

1- Présentation de la carte ESP8266

2- Installation et prise en main de l'IDE Visual Studio Code (VSC)

- **Test carte ESP8266 avec sketches ESP8266Sketch1**

3 - Etude de la fonction millis()

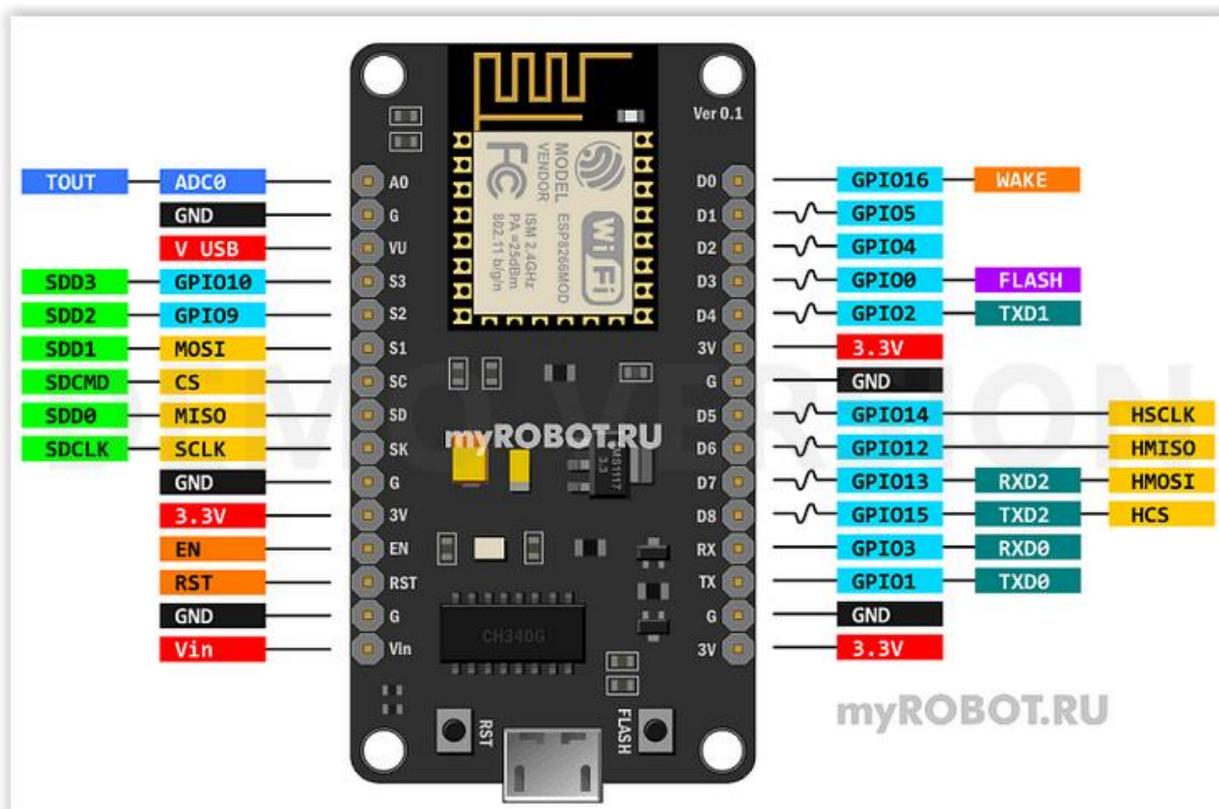
4- Etude des sketches :

- **ESP8266Sketch2_InfoSysteme**
- **ESP8266Sketch3_ConnexionWiFi**
- **ESP8266Sketch4_ConnexionMultiWiFi**
- **ESP8266Sketch5_ConnexionWiFiAdrIP_Fixe**
- **ESP8266Sketch6_ServeurWeb1**
- **ESP8266Sketch7_ServeurWeb2**
- **ESP8266Sketch8_PageWebDynamique**

5 – Les Interruptions

6 - Projet : Station Météo Connectée

1 - Présentation ESP8266 NodeMCU LUA CH340 ESP-12E



- Les GPIO6 à GPIO11 sont utilisés par la mémoire Flash interne de l'ESP8266, ils ne sont pas disponibles pour une utilisation générale (par exemple pour connecter des capteurs ou des LEDs).
- GPIO16 (D0) est différent des autres GPIO et ne prend pas en charge le PWM ni les interruptions. Il est souvent utilisé pour réveiller le module ESP8266 de son mode Deep Sleep.
- Les broches RX (GPIO3) et TX (GPIO1) sont utilisées pour la communication UART (série), et il est recommandé de les laisser libres pour le téléchargement du code ou la communication série.
- Les broches les plus sécurisées pour une utilisation en entrée/sortie générale sont : D1 (GPIO5), D2 (GPIO4), D5 (GPIO14), D6 (GPIO12), D7 (GPIO13).

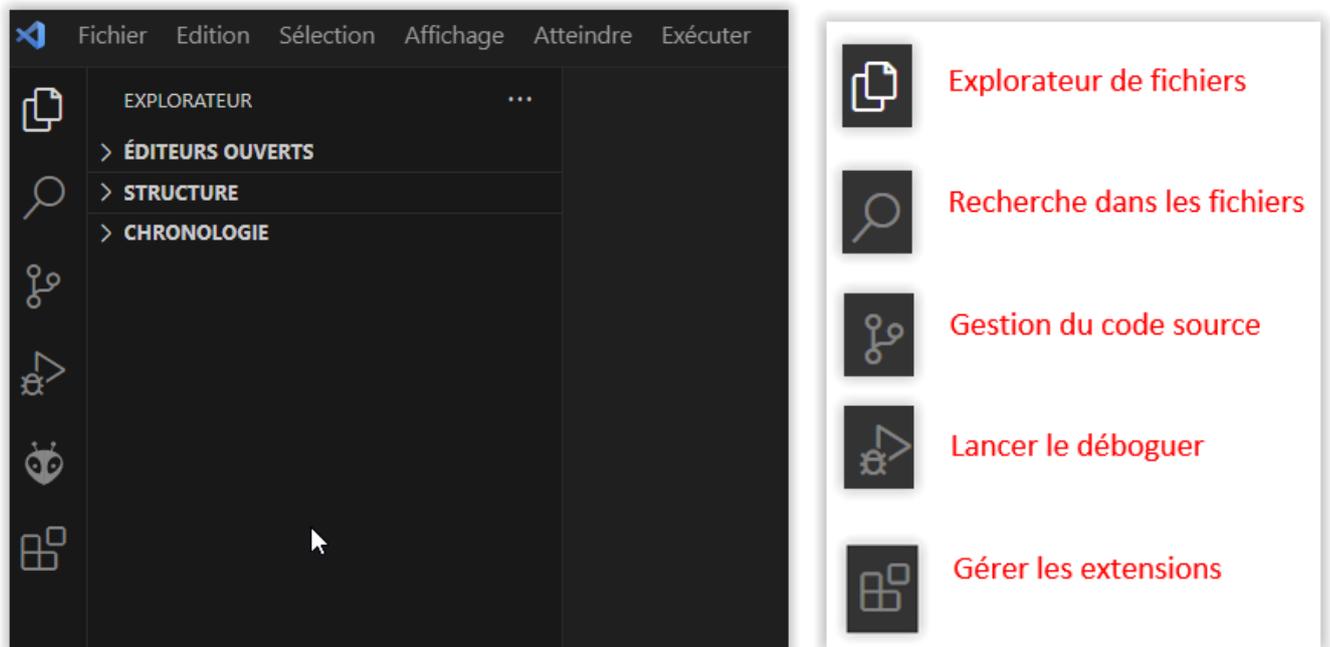
Pin Nom	GPIO	Fonction	Notes
D0	GPIO16	Wake (Réveil)	Pas de PWM, ni interruptions
D1 	GPIO5	SCL (I2C), PWM	Prend en charge les interruptions et PWM
D2 	GPIO4	SDA (I2C), PWM	Prend en charge les interruptions et PWM
D3	GPIO0	Flash Button, PWM	Utilisé aussi pour boot mode, Attention
D4	GPIO2	TX1, PWM, LED_BUILTIN	Utilisé aussi pour boot mode, Attention
D5 	GPIO14	SCLK (SPI), PWM	Prend en charge les interruptions et PWM
D6 	GPIO12	MISO (SPI), PWM	Prend en charge les interruptions et PWM
D7 	GPIO13	MOSI (SPI), PWM	Prend en charge les interruptions et PWM
D8	GPIO15	CS (SPI), PWM	Nécessite une résistance pull-down pour garantir un bon démarrage
RX	GPIO3	RXD0 (UART)	Utilisé pour la communication série, éviter de l'utiliser pour GPIO
TX	GPIO1	TXD0 (UART)	Utilisé pour la communication série, éviter de l'utiliser pour GPIO
A0	A0	Entrée analogique	Lecture de 0 à 1V seulement
3V3	-	Alimentation 3.3V	Alimentation du module
GND	-	Masse	Connexion à la terre / GND
EN	-	Enable	Active le module ESP8266
RST	-	Reset	Réinitialisation du module
-	GPIO6	Flash Memory (SCK)	Non accessible, réservé pour la mémoire Flash interne
-	GPIO7	Flash Memory (SD0)	Non accessible, réservé pour la mémoire Flash interne
-	GPIO8	Flash Memory (SD1)	Non accessible, réservé pour la mémoire Flash interne
-	GPIO9	Flash Memory (SD2)	Non accessible, réservé pour la mémoire Flash interne
-	GPIO10	Flash Memory (SD3)	Non accessible, réservé pour la mémoire Flash interne
-	GPIO11	Flash Memory (CMD)	Non accessible, réservé pour la mémoire Flash interne

2 - Installation et prise en main de l'IDE Visual Studio Code (VSC) :

- Téléchargement et installation de Visual Studio Code (VSC) :

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

- Découverte et paramétrage de VSC pour Arduino :



Module linguistique français pour VSC

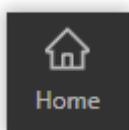
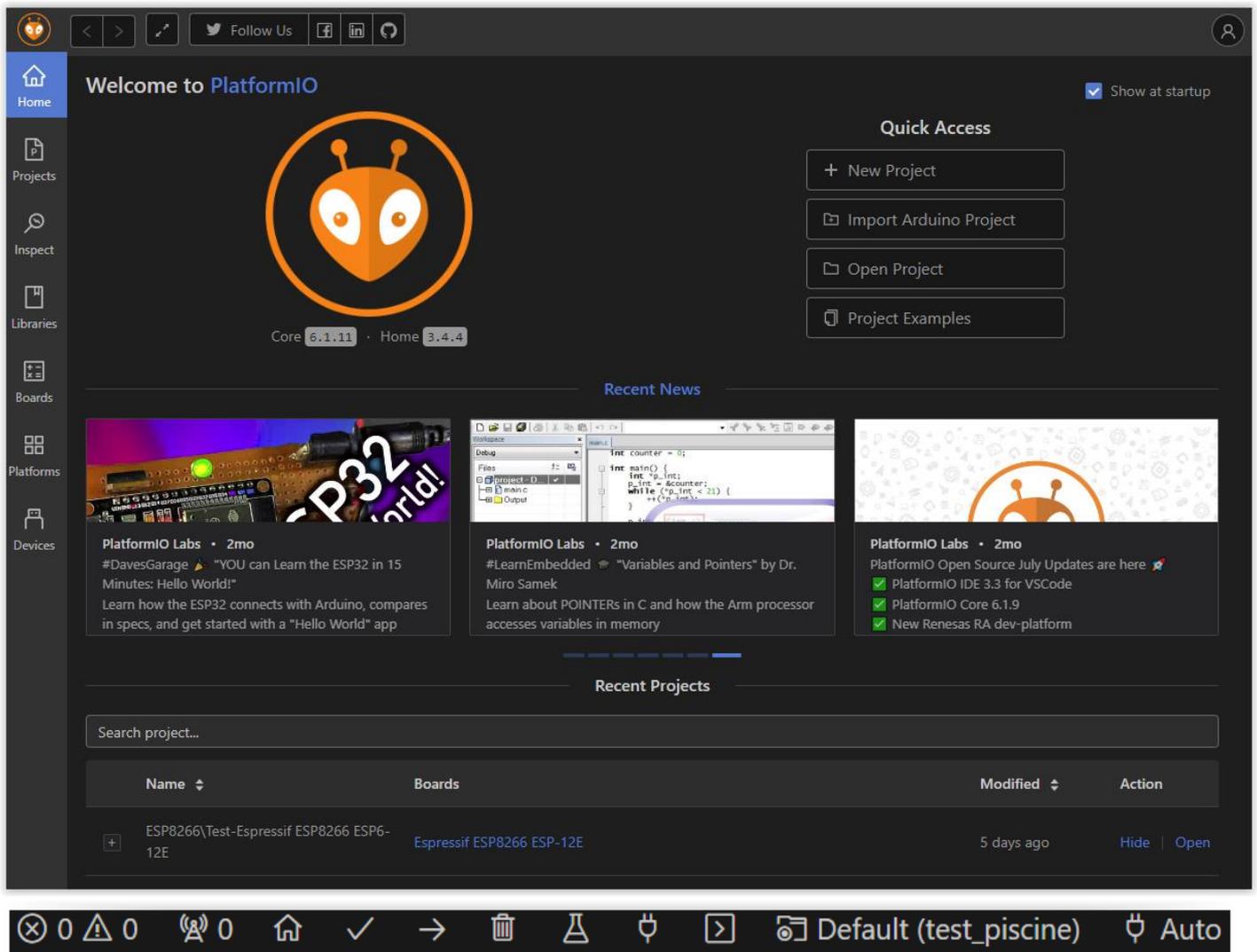


Extention PlatformIO pour Arduino

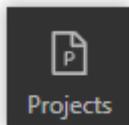


Live serveur

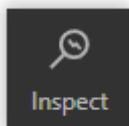
- Présentation de PlatformIO



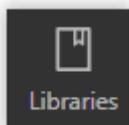
Affiche la page de "Bienvenue" de PlatformIO



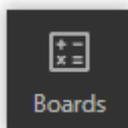
Affiche les projets sous PlatformIO



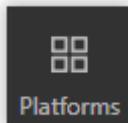
Contrôle les ressources utilisées par votre projet



Affiche les librairies



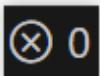
Affiche cartes reconnues par PlatformIO



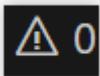
Gère les cartes installées



Affiche le **port Com** utilisé



Affiche le nombre d'erreur dans l'écriture du code



Affiche le nombre de tâche en fonctionnement



Affiche la page de "**Bienvenue**" de PlatformIO



Exécute la vérification et la construction du code



Téléverse le code dans la carte



Efface le terminal



Teste le code



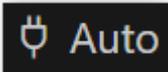
Affiche le moniteur série



Ouvre un nouveau terminal



Default (test_piscine) Affiche le projet en cours



Détection du port automatique

Exercice :

Faire clignoter la LED intégrée à la carte ESP8266 toutes les 500 ms.

3 - Etude de la fonction millis()

```
const byte led = 12;
unsigned long tempsPrecedent = 0;
const long intervalle = 500;
void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  unsigned long tempsActuel = millis();

  if (tempsActuel - tempsPrecedent >= intervalle) {
    tempsPrecedent = tempsActuel;
    digitalWrite(led, !digitalRead(led));
  }
}
```

Ce programme permet de faire clignoter une LED connectée à une carte ESP8266 de manière régulière, en utilisant un intervalle de temps défini (500 millisecondes dans cet exemple).

Le programme utilise la fonction millis() plutôt que delay(), permettant ainsi à d'autres tâches de s'exécuter en parallèle, une approche idéale pour les systèmes plus complexes.

```
const byte led = 12;
unsigned long tempsPrecedent = 0;
const long intervalle = 500;
```

- **const byte led = 12**

Cette ligne déclare une variable constante led de type byte, qui stocke la broche du microcontrôleur à laquelle la LED est connectée. Le numéro 12 indique que la LED est connectée à la broche numérique 12 de l'ESP8266.

- **unsigned long tempsPrecedent = 0**

La variable tempsPrecedent est utilisée pour mémoriser l'instant (en millisecondes) où la dernière action a été effectuée (comme changer

l'état de la LED). Cette variable est de type unsigned long car la fonction millis() retourne également une valeur de ce type.

- **const long intervalle = 500**

Cette constante déclare un intervalle de temps de 500 millisecondes (soit 0,5 seconde). Cet intervalle détermine à quelle fréquence la LED changera d'état (s'allumera ou s'éteindra).

Ici, 500 ms signifie que la LED clignotera à une fréquence de 2 fois par seconde.

```
void setup() {  
  pinMode(led, OUTPUT);  
}
```

- **pinMode(led, OUTPUT)**

Cette commande configure la broche 12 (référéncée par led) en tant que sortie (OUTPUT). Cela signifie que cette broche pourra envoyer un signal de tension pour contrôler la LED (allumer ou éteindre).

```
unsigned long tempsActuel = millis();
```

- **millis()**

Cette fonction retourne le nombre de millisecondes écoulées depuis le démarrage du microcontrôleur. C'est une horloge interne très utile pour mesurer des intervalles de temps sans bloquer l'exécution du programme (contrairement à delay()).

- **tempsActuel**

Cette variable enregistre l'heure actuelle en millisecondes, obtenue grâce à millis().

```
if (tempsActuel - tempsPrecedent >= intervalle) {
```

- **Vérification de l'intervalle de temps écoulé :**

La condition vérifie si la différence entre l'heure actuelle (tempsActuel) et l'heure mémorisée lors de la dernière action (tempsPrecedent) est supérieure ou égale à l'intervalle défini (500 millisecondes).

- **But de cette vérification :**

Cela permet de s'assurer que 500 ms se sont écoulées avant de changer l'état de la LED. Si cette condition est vraie, cela signifie que 500 ms sont passées et qu'il est temps de changer l'état de la LED.

```
tempsPrecedent = tempsActuel;
```

- Une fois l'intervalle écoulé, on met à jour la variable tempsPrecedent avec la valeur de tempsActuel, de manière à mémoriser le moment où l'action a été effectuée. Cela permet de recommencer la vérification pour le prochain cycle.

```
digitalWrite(led, !digitalRead(led));
```

- **digitalRead(led) :**

Cette fonction lit l'état actuel de la broche led. Si la LED est allumée, cela renvoie HIGH (ou 1), et si elle est éteinte, cela renvoie LOW (ou 0).

- **!digitalRead(led) :**

L'opérateur logique ! inverse l'état actuel de la LED. Donc, si la LED est allumée (HIGH), elle sera éteinte (LOW), et si elle est éteinte, elle sera allumée.

- **digitalWrite(led, !digitalRead(led)) :**

Cette ligne change l'état de la LED en fonction de son état actuel. Si la LED est allumée, elle sera éteinte, et vice versa.

Objectif de l'exercice :

L'objectif de cet exercice est d'apprendre à utiliser la fonction **millis()** pour contrôler le clignotement de deux LEDs à des fréquences différentes.

Contrairement à la fonction **delay()**, qui bloque l'exécution du programme, **millis()** permet de mesurer le temps écoulé sans bloquer le reste du programme, ce qui est très utile pour faire plusieurs tâches en parallèle.

Concepts clés :

1. **millis()** : Cette fonction retourne le temps écoulé en millisecondes depuis le démarrage de l'ESP8266. Nous allons l'utiliser pour vérifier si un certain intervalle de temps s'est écoulé avant de changer l'état des LEDs.
2. **Deux fréquences différentes** : Chaque LED aura un temps de clignotement différent, ce qui signifie que chaque LED changera d'état (ON/OFF) selon un intervalle unique.

Étapes :

1. **Connexion des LEDs** : Connectez deux LEDs aux broches GPIO de l'ESP8266 (par exemple, GPIO 4 et GPIO 5), avec des résistances pour protéger les LEDs.
2. **Gestion des intervalles avec millis()** : Créez deux variables pour stocker les intervalles de temps souhaités pour chaque LED (par exemple, 200 ms pour une LED et 2000 ms pour l'autre).
3. **Comparaison avec millis()** : Utilisez la fonction **millis()** pour vérifier si l'intervalle de temps est dépassé, et changez l'état des LEDs lorsque c'est le cas.

Exemple d'application :

- La première LED clignotera toutes les 200 millisecondes (1 clignotement toutes les 0,2 secondes).
- La deuxième LED clignotera toutes les 2000 millisecondes (1 clignotement toutes les 2 secondes).