Arduino IDE
et ESP32forth

Marc PETREMANN

Version 1.1 - 24/01/26

Table des matières

Préambule

Ce manuel est destiné à comprendre l'utilisation du programme ARDUINO IDE pour compiler ESP32forth pour cartes EPS32 et accessoirement tous les autres programmes en langage C pour la gamme des cartes de l'environnement ARDUINO.

Installer ARDUINO IDE

Téléchargez la version ARDUINO IDE la plus récente, disponible sur le site officiel ;
https://www.arduino.cc/en/software/

Sélectionnez le système cible, puis cliquez sur *Download*.

Une fois téléchargé, suivez les instructions spécifiques à votre système (Windows, Linux, MacOS…).

Installation sous Windows

Ici, la version téléchargée a comme nom de fichier *arduino-ide_2.3.7_Windows_64bit.exe*.

Après téléchargement, le fichier doit se trouver dans le dossier *Téléchargements*.

Pour lancer l'installation, lancez l'exécution de ce fichier. Suivez les différentes étapes demandées lors de l'installation.

ARDUINO IDE

Dans le contexte d'Arduino, le sigle **IDE** signifie **Integrated Development Environment**, ce qui se traduit en français par **Environnement de Développement Intégré**.

C'est l'outil indispensable qui sert de "centre de commande" pour créer vos projets électroniques.

L'IDE est un logiciel unique qui regroupe tous les outils dont un programmateur a besoin pour transformer une idée en code, puis en action sur une carte Arduino. Il se compose principalement de trois parties :

- **L'Éditeur de texte** : C'est là que vous écrivez votre code (appelé "sketch" chez Arduino). Il colore les mots-clés pour vous aider à ne pas faire d'erreurs de syntaxe.
- **Le Compilateur** : Il traduit votre code (compréhensible par l'humain) en langage machine (des 0 et des 1) que le microcontrôleur de la carte peut comprendre.
- **Le Programmateur (Uploader)** : Il envoie le code compilé vers votre carte Arduino via le câble USB.

Avant l'existence des IDE, les développeurs devaient utiliser des logiciels séparés : un pour écrire, un pour compiler, et un autre pour envoyer le fichier sur la puce. L'Arduino IDE intègre tout cela dans une interface simple, ce qui rend l'électronique accessible même aux débutants.

Les deux versions principales :

- Arduino IDE 1.8.x : La version classique, très stable et légère.
- Arduino IDE 2.x : La version moderne avec l'autocomplétion (il devine ce que vous allez écrire) et un débogueur intégré.

Pour compiler et télécharger ESP32forth, il vous faudra la version 2.x.

Télécharger et préparer ESP32forth

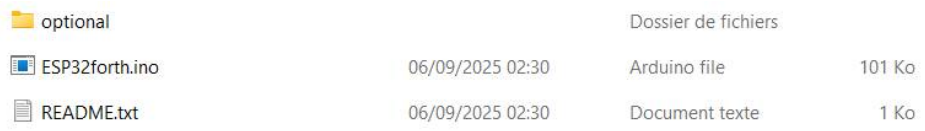
Pour télécharger ESP32forth, allez ici :
https://esp32forth.appspot.com/ESP32forth.html

Préférez la version 7.0.7.21. C'est la version compatible avec ARDUINO IDE 2.x.

Préparation des dossiers pour ESP32forth

Ouvrez le fichier Zip contenant ESP32forth et décompressez son contenu dans votre espace de travail de développement. Il est préférable de choisir un espace de travail ailleurs que sur le bureau Windows.

Une fois décompressé, vous devez avoir un dossier ESP32forth avec ces fichiers :

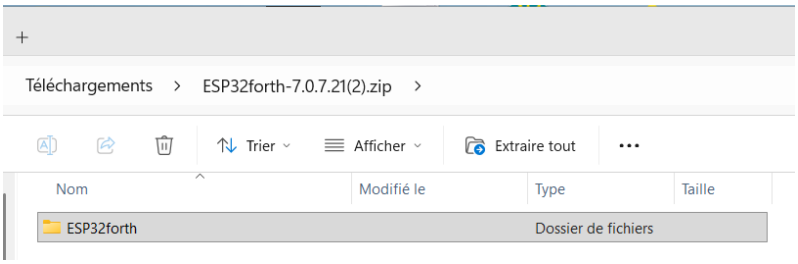


Si vous n'avez jamais compilé et installé ESP32forth, laissez le contenu de ces dossiers en l'état.

Si vous n'avez jamais développé pour Arduino, ESP32 ou tout autre carte électronique, créez un dossier **developpements** dans votre espace utilisateur de windows :

`c : → Utilisateurs → moiMeme → developpements → ESP32`

Dans ce dossier **developpements**, créez un autre dossier **ESP32**. C'est dans ce dossier que vous copiez le contenu du fichier Zip précédemment téléchargé :

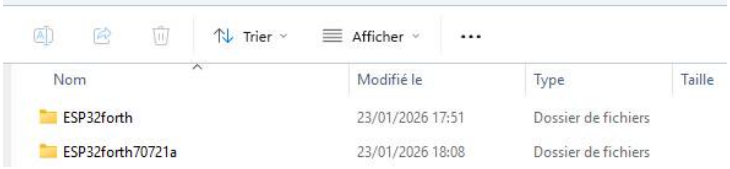


Si vous avez d'autres codes à créer pour ESP32, vous définirez d'autres répertoires dans le dossier **ESP32**.

Ouvrir un projet

Ouvrez l'IDE ARDUINO. Puis ouvrez le projet précédemment téléchargé. Pour ce faire, cliquez sur *File* et sélectionnez *Open*. Recherchez le projet ESP32forth. Sélectionnez le fichier **ESP32forth.ino**. Vous arrivez sur cette fenêtre :

ATTENTION : le projet doit avoir le même nom que le dossier qui le contient. Par exemple, vous voulez faire une copie de ce projet pour y apporter des modifications. Créez un nouveau dossier **ESP32forth70721a** et mettez la copie de **ESP32forth.ino** dans ce nouveau dossier. Renommez ensuite **ESP32forth** en **ESP32forth70721a** :



Vous pourrez faire des modifications dans ce nouveau dossier sans altérer le contenu du dossier **ESP32forth**.

A ce stade, il faut gérer quelques paramètres avant de téléverser notre projet vers la carte ESP32.

Sélection d’une carte spécifique

Avant de voir la liste des cartes, l'IDE doit savoir ce qu'est un ESP32.

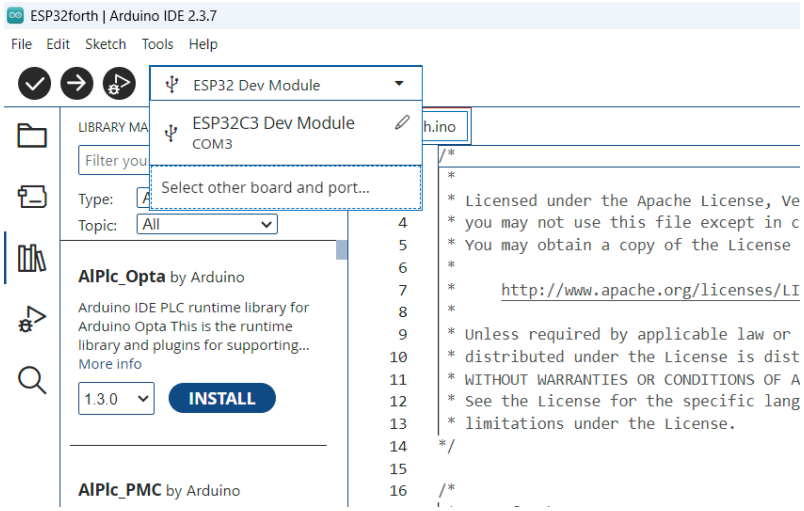
- Allez dans File **> Preferences**.
- Dans le champ **Additionnal boards manager URLs**, collez ce lien : `https://dl.espressif.com/dl/package_esp32_index.json`
- Allez ensuite dans **Tools > Board > Boards Manager**, recherchez **ESP32** et cliquez sur **Instal**.

Une fois le pack installé, voici comment choisir votre module :

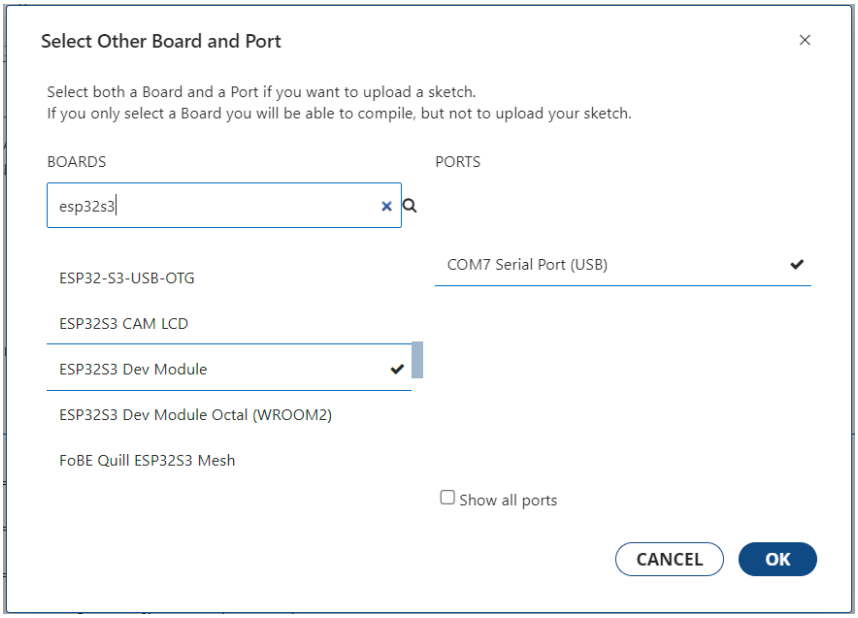
1. Ouvrez le menu **Tools**.
2. Placez votre souris sur **Board : "..."** (souvent "Arduino Uno" par défaut).
3. Cherchez la section **esp32**. Un sous-menu s'ouvrira avec une liste impressionnante de modèles.
4. Choisissez votre modèle :
    - Pour un module classique à 30 ou 38 broches, choisissez **DOIT ESP32 DEVKIT V1**.
    - Si vous avez une version plus récente (S2, S3 ou C3), cherchez le nom correspondant (ex: **ESP32S3 Dev Module**).

Les modèles ne sont pas classés par ordre alphabétique, mais par marques ou modèles.

L’autre solution, plus aisée, consiste à passer par ce sélecteur qui se situe en dessous de *tools* :



Dans la liste déroulante, sélectionnez Select *other board and port.* Dans la fenêtre pop-up, on recherche notre carte et le port USB sur laquelle elle est raccordée :



Ici, nous avons sélectionné la carte **ESP32S3 Dev Module** et le port **COM7**. Le port COM peut être différent pour vous.

Les outils

Les outils sont accessibles en cliquant sur *Tools* dans la barre de menu. Ces outils sont très nombreux :

```
Auto Format
Archive Sketch
Manage Libraries
Serial Monitor
Serial Plotter
--------------------------------
Firmware Updater
Upload SSL Root Certification
--------------------------------
Board:
Port:
Reload Bord Data
Get Board Info
--------------------------------
CPU Frequency:
Core Debug Level:
Erase All Flash Before Sketch Upload:
Events Run On:
Flash Frequency:
Flash Mode:
Flash Size:
JTAG Adapter:
Arduino Runs On:
Partition Scheme:
PSRAM:
Upload Speed:
ZigBee Mode:
--------------------------------
Programmer
Burn Bootloader
```

Notre propos n’est pas de détailler tous ces outils. Nous allons nous intéresser en priorité aux paramètres essentiels à notre projet **ESP32forth**.

Auto Format

La fonction *Auto Format* est le meilleur ami des développeurs qui aiment le travail propre (ou de ceux qui sont un peu désordonnés).

Concrètement, elle sert à réorganiser automatiquement la mise en page de votre code pour le rendre parfaitement lisible, sans en changer le fonctionnement.

En programmation, la lisibilité est cruciale. L'Auto Format intervient sur plusieurs points :

- **L'indentation** : Il aligne correctement les retraits à gauche. Par exemple, tout ce qui se trouve à l'intérieur d'une fonction `void loop() { ... }` sera décalé d'un cran pour montrer visuellement la hiérarchie.
- **Les accolades** : Il replace les `{` et `}` selon des standards de programmation précis, évitant ainsi de s'emmêler les pinceaux dans les structures de contrôle.
- **Les espaces** : Il ajoute des espaces là où ils manquent (par exemple autour d'un signe = ou après une virgule) pour aérer le texte.

Archive Sketch

L'option *Archive Sketch* est une fonctionnalité très pratique et pourtant souvent ignorée dans l'Arduino IDE. Elle sert à créer instantanément une copie de sauvegarde compressée de votre projet actuel.

Lorsqu'on travaille sur un projet électronique, on accumule souvent plusieurs fichiers (le code principal, des fichiers d'en-tête .h, des onglets secondaires). Cette option permet de tout "emballer" proprement.

Ce que fait concrètement *Archive Sketch* :

- **Compression en .zip** : Elle prend le dossier complet de votre projet et le transforme en une archive compressée.
- **Horodatage automatique** : L'IDE nomme automatiquement le fichier avec la date du jour (par exemple : mon_projet_jan23a.zip). Cela permet de créer des points de restauration sans écraser vos versions précédentes.
- **Organisation** : Le fichier compressé est placé directement dans le dossier de votre projet, prêt à être déplacé.

L'utilité réelle :

- **Partage facilité** : Si vous voulez envoyer votre code à un ami ou à un professeur, envoyer un seul fichier *.zip* est bien plus simple que d'envoyer tout un dossier avec plusieurs fichiers.
- **Historique de version** (Sauvegarde) : Avant de tenter une modification risquée qui pourrait tout "casser", un petit coup d'*Archive Sketch* vous permet de garder une copie de l'état actuel qui fonctionne.
- **Nettoyage** : Cela permet de stocker vos vieux projets de manière compacte sur votre ordinateur ou sur un cloud.

Manage Libraries

L'option **Manage Libraries** (Gérer les bibliothèques) est l'un des outils les plus puissants de l'Arduino IDE. Elle permet d'étendre les capacités de votre carte sans que vous ayez à réinventer la roue.

En informatique, une **bibliothèque** (ou *library*) est un ensemble de codes déjà écrits par des professionnels ou la communauté pour faire fonctionner un composant spécifique (un écran LCD, un capteur de température, un moteur, etc.).

C'est une sorte de "magasin d'applications" gratuit pour votre code. Au lieu de configurer manuellement chaque broche d'un écran complexe, vous installez une bibliothèque qui vous offre des commandes simples comme `ecran.ecrire("Bonjour")`.

Le gestionnaire sert à :

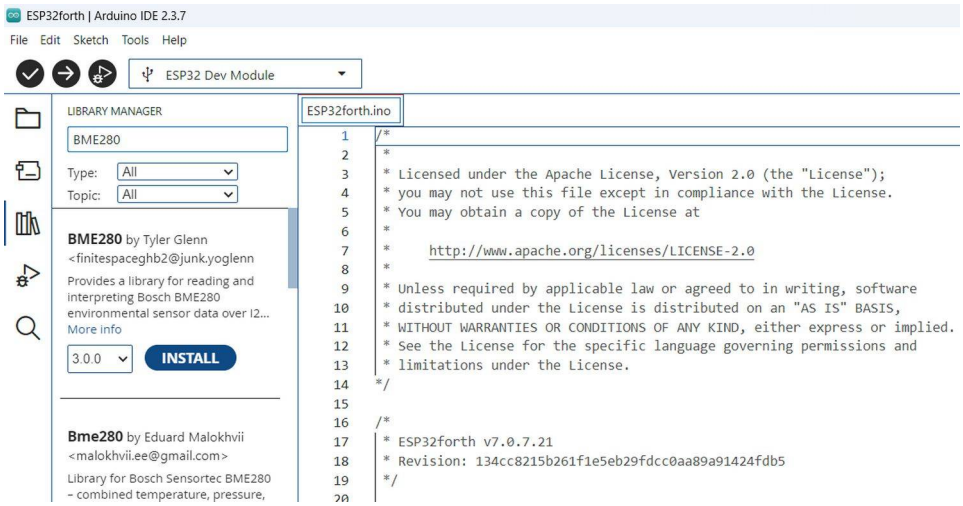
- **Rechercher** des bibliothèques par nom ou par fonctionnalité.
- **Installer** de nouvelles bibliothèques en un clic.
- **Mettre à jour** vos bibliothèques pour corriger des bugs ou ajouter des fonctions.
- **Gérer les versions** (si une mise à jour casse votre code, vous pouvez revenir à une version plus ancienne).

Utilisation :

1. **Ouverture :** Allez dans **Outils > Manage Libraries** ou cliquez sur l'icône de "livres" dans la barre latérale gauche (en version 2.x).

2. **Recherche :** Tapez le nom d'un composant (ex: "DHT11" pour un capteur d'humidité).
3. **Installation :** Cliquez sur le bouton "Install". L'IDE télécharge et installe tout au bon endroit pour vous.

Exemple concret, vous achetez un capteur de pression **BME280**, vous ne saurez probablement pas comment interpréter ses signaux électriques bruts. En allant dans *Manage Libraries*, vous cherchez "BME280", vous installez celle d'Adafruit, et soudain, votre Arduino comprend la commande `capteur.readPressure()`.

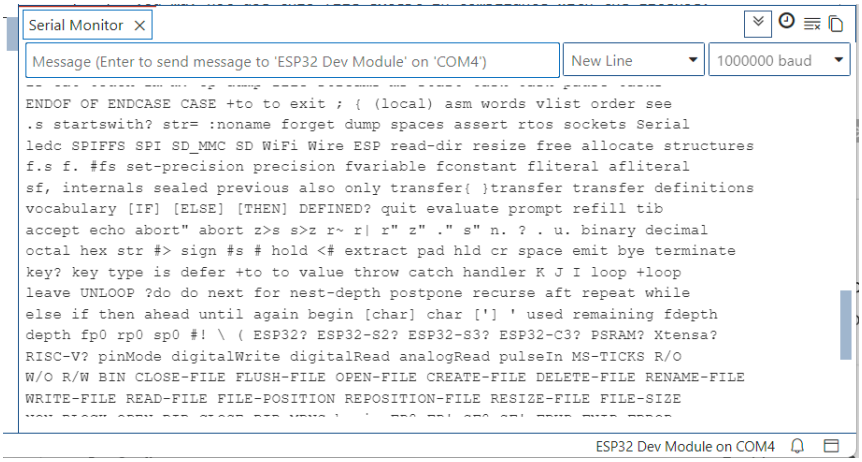


Dans la copie écran ci-dessus, si vous cliquez sur *INSTALL*, la librairie **BME280** sera installée. Mais tant que votre projet ne fait pas explicitement appel aux fonctions de cette librairie, cette installation n'aura aucun impact sur la taille du code binaire généré lors de la compilation du code applicatif.

Serial Monitor

Le **Serial Monitor** (Moniteur Série) est l'outil de diagnostic essentiel intégré à l'IDE. C'est le canal de communication qui permet à votre carte Arduino de "parler" à votre ordinateur en temps réel.

Puisqu'une carte ESP32 n'a pas d'écran par défaut, *Serial Monitor* sert de fenêtre sur ce qui se passe à l'intérieur du microcontrôleur :



Ici, on a exécuté `page vlist` dans cette fenêtre de terminal.

Pour que le texte s'affiche correctement, la vitesse de communication (le **Baud Rate**) réglée dans le Moniteur Série doit être **identique** à celle spécifiée dans votre code (généralement **9600** ou **115200**). Si les vitesses ne correspondent pas, vous verrez des caractères bizarres ou illisibles (du "gibberish").

Board

L'option **Board** (Type de carte) dans l'Arduino IDE est le réglage le plus crucial avant de téléverser votre code. Elle indique au logiciel exactement à quel type de matériel il s'adresse.

Comme chaque puce a un cerveau différent (nombre de pattes, vitesse de calcul, mémoire), l'IDE doit savoir s'il prépare le code pour une petite Arduino Uno ou pour une bête de course comme l'**ESP32**.

Importance pour l'ESP32

Contrairement à Arduino qui fabrique ses propres cartes, l'**ESP32** est un microcontrôleur produit par la société *Espressif*, mais utilisé par des dizaines de fabricants différents (DOIT, Adafruit, LilyGO, Heltec...).

Chaque carte a une disposition de broches (pinout) différente. Si vous choisissez la mauvaise "Board", vous risquez :

- Que le code ne se téléverse pas du tout.
- Que vos branchements ne correspondent pas (ex: vous demandez d'allumer la broche 2, mais sur votre modèle spécifique, c'est la broche 4).

Exemples de cartes ESP32 courantes

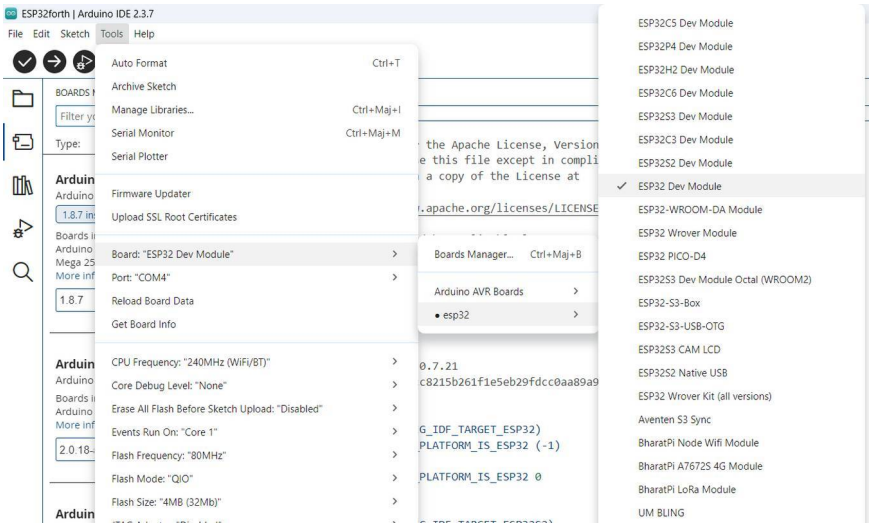
Lorsque vous allez dans *Tools → Board → ESP32*, vous verrez une liste immense. Voici les plus fréquentes :

Nom dans l'IDE	Usage typique
DOIT ESP32 DEVKIT V1	La version la plus classique et bon marché (30 broches). Idéale pour débuter.
Adafruit Feather ESP32	Format compact avec connecteur de batterie intégré.
ESP32 Dev Module	Le réglage "universel". Si vous ne trouvez pas votre modèle exact, celui-ci fonctionne souvent par défaut.
Wemos D1 Mini ESP32	Une version miniature pour les projets où l'espace est réduit.

Configuration d’une carte ESP32

Par défaut, l'Arduino IDE ne connaît pas l'ESP32. Il faut d'abord "l'éduquer" :

1. **Ajout de l'URL :** Dans les *Préférences*, on ajoute le lien du gestionnaire de cartes Espressif.
2. **Installation :** Dans *Tools → Board → Boards Manager*, on cherche "ESP32" et on installe le paquet.
3. **Sélection :** Une fois installé, vous allez dans *Tools → Board → ESP32* et vous sélectionnez votre modèle précis (ex: *DOIT ESP32 DEVKIT V1*).



Attention : Une fois la carte choisie, vérifiez aussi le port (COM) dans le menu *Tools*, sinon votre ordinateur ne saura pas par quel "tuyau" envoyer le code !

Port

Cet outil permet de sélectionner le port série sur lequel est connectée la carte ESP32 à programmer.

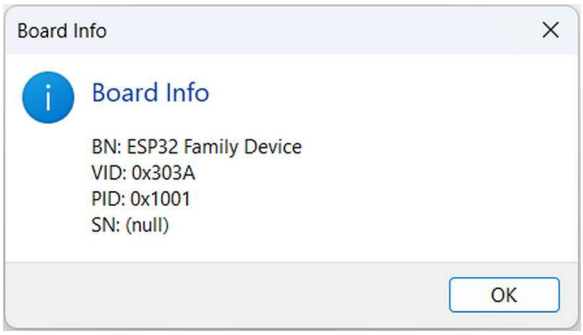
ARDUINO IDE ne propose que les ports actifs ou récemment utilisés. Si votre carte ESP32 est connectée à un port USB, l'IDE peut détecter cette carte et indique le port COM associé.

Get Board Info

L'option **Get Board Info** (Obtenir les informations de la carte) est une sorte de "carte d'identité numérique" qui interroge le matériel branché sur votre port USB.

C'est l'outil parfait pour vérifier si votre ordinateur "voit" réellement l'Arduino et pour identifier précisément quel composant électronique assure la communication.

Cette option ouvre une petite fenêtre qui affiche des informations techniques brutes envoyées par la puce USB de la carte. C'est utile pour :



1. **Vérifier la connexion :** Si la fenêtre affiche une erreur, cela signifie que le câble est défectueux ou que les pilotes (drivers) ne sont pas installés.
2. **Identifier un clone :** Si vous avez acheté une carte "compatible Arduino", cette option vous dira quel composant est utilisé pour la communication (souvent une puce différente de l'originale).
3. **Trouver le port :** Si vous avez plusieurs périphériques branchés, cela vous confirme que vous avez sélectionné le bon port COM.

La fenêtre affiche généralement trois codes clés :

- **BN (Board Name) :** Le nom de la carte (si elle est reconnue officiellement).
- **VID (Vendor ID) :** L'identifiant du fabricant du composant USB.
- **PID (Product ID) :** L'identifiant du modèle de produit.

- **SN (Serial Number) :** Le numéro de série unique de la puce (si disponible).

Il est fréquent que pour un **ESP32** ou une **Arduino Nano générique**, le message affiche **"Unknown Board"** (Carte inconnue).

> **Pourquoi ?** > Parce que ces cartes utilisent souvent des puces USB génériques (comme le **CH340** ou le **CP2102**). L'IDE reconnaît qu'il y a "quelque chose" au bout du fil (le VID/PID s'affiche), mais il ne peut pas garantir qu'il s'agit d'une carte Arduino officielle.

Ce n'est pas grave : tant que vous voyez un numéro de VID et de PID, cela signifie que la communication fonctionne. Vous pouvez ignorer le "Unknown Board" et téléverser votre code normalement.

CPU Frequency

L'option *CPU Frequency* (Fréquence du processeur) détermine la "vitesse de réflexion" de votre microcontrôleur. C'est le rythme cardiaque de la puce, cadencé par un oscillateur.

Plus la fréquence est élevée, plus l'ESP32 exécute d'instructions par seconde.

Sur une carte Arduino Uno classique, la fréquence est fixe (16 MHz). Mais sur un **ESP32**, vous avez souvent le choix (généralement entre **80 MHz, 160 MHz et 240 MHz**). Modifier ce réglage sert principalement à deux choses :

1. La Performance (Puissance de calcul)

Si vous faites du traitement de signal, de l'affichage vidéo ou du calcul mathématique complexe, vous choisirez la fréquence maximale (**240 MHz**). Le processeur traitera les données beaucoup plus rapidement.

2. L'Autonomie (Économie d'énergie)

C'est l'usage le plus courant pour baisser la fréquence.

- **Plus la fréquence est haute**, plus la puce consomme de courant et chauffe.
- **Plus la fréquence est basse**, plus vous économisez votre batterie.

> Si votre projet consiste juste à lire une température une fois par minute, faire tourner le processeur à 240 MHz est un gaspillage d'énergie. Passer à 80 MHz peut prolonger la vie de votre batterie de manière significative.

Les conséquences d'un changement de fréquence

Attention, changer la fréquence n'est pas "gratuit" techniquement :

- **Le WiFi et le Bluetooth :** Sur l'ESP32, certaines fonctions sans fil nécessitent une fréquence minimale (souvent 80 MHz ou plus) pour fonctionner correctement. Si vous descendez trop bas, le WiFi se coupera.
- **Les timings :** Bien que l'IDE gère cela la plupart du temps, certains calculs de temps (comme les délais ou les protocoles de communication très précis) peuvent être perturbés si vous changez la fréquence en plein milieu d'un projet.

Résumé des réglages types :

Fréquence	Usage idéal
240 MHz	WiFi actif, Bluetooth, calculs intensifs, écrans tactiles.
160 MHz	Bon compromis puissance / consommation.
80 MHz	Projets simples, capteurs IoT sur batterie, WiFi occasionnel.

Fréquence	Usage idéal
< 80 MHz	Mode très basse consommation (Deep Sleep), pas de sans-fil.

Core Debug Level

L'option **Core Debug Level** (Niveau de débogage du noyau) est une fonctionnalité spécifique aux cartes puissantes comme l'**ESP32**. Elle permet de choisir la quantité de détails techniques que la carte envoie vers le **Serial Monitor** concernant son fonctionnement interne.

C'est un peu comme si vous demandiez à votre voiture de vous dire soit "Tout va bien", soit de vous lister chaque petite goutte d'essence injectée dans le moteur.

À quoi servent les différents niveaux ?

Le "Core" (le noyau) de l'ESP32 gère des tâches complexes en arrière-plan (connexion WiFi, gestion de la mémoire, Bluetooth). Si votre code plante sans raison apparente, augmenter le niveau de debug permet de voir l'erreur exacte que le système a rencontrée.

Voici les réglages disponibles (du moins bavard au plus verbeux) :

- **None (Aucun) :** C'est le réglage par défaut pour la production. La carte reste silencieuse, ce qui économise de la mémoire et de l'énergie.
- **Error (Erreur) :** La carte ne parle que si quelque chose de grave se produit (un crash, un composant défectueux).
- **Warn (Avertissement) :** Affiche les erreurs, mais aussi des messages de prévention sur des comportements suspects qui n'ont pas encore fait planter le système.
- **Info (Information) :** Affiche les étapes clés, comme "Connexion WiFi réussie" ou "Adresse IP obtenue". Très utile pour suivre le déroulement normal.
- **Debug :** Fournit des détails techniques sur les fonctions internes. Utile pour les développeurs avancés.
- **Verbose (Verbeux) :** La carte raconte *tout* ce qu'elle fait, à chaque milliseconde. Attention : cela ralentit légèrement l'exécution et inonde votre Moniteur Série de texte.

Pourquoi l'utiliser ?

1. **Traquer les "Gurus" (Crashes) :** L'ESP32 affiche souvent un message d'erreur appelé "Guru Meditation Error" quand il plante. Avec un niveau de debug sur **Info** ou **Debug**, vous obtiendrez plus de détails sur la raison du crash (mémoire pleine, division par zéro, etc.).
2. **Surveiller le WiFi :** Si vous n'arrivez pas à vous connecter à votre box, mettre le niveau sur **Info** vous permettra de voir si c'est un problème de mot de passe ou de signal trop faible.

Conseil pratique : Laissez cette option sur **None** ou **Error** la plupart du temps. N'utilisez **Info** ou **Verbose** que pendant que vous êtes en train de chercher la cause d'un bug précis.

Erase All Flash Before Sketch Upload

L'option **"Erase All Flash Before Sketch Upload"** (Effacer toute la mémoire Flash avant le téléversement) est une commande de "nettoyage profond" pour votre ESP32.

Par défaut, quand vous envoyez un nouveau programme (sketch) sur votre carte, l'IDE remplace uniquement la zone de mémoire nécessaire au code. Le reste de la mémoire Flash (là où sont stockés vos réglages WiFi, vos fichiers SPIFFS/LittleFS ou vos données sauvegardées) reste intact.

En activant cette option, vous demandez à l'IDE de **formater intégralement** la puce avant d'écrire le nouveau code.

Il n'est pas nécessaire (ni recommandé) de l'utiliser à chaque fois, car cela ralentit le processus de téléversement. Voici les cas où elle devient indispensable :

- **Comportement erratique :** Votre code est correct, mais la carte agit bizarrement, redémarre en boucle ou garde en mémoire d'anciens paramètres WiFi dont vous ne voulez plus.
- **Changement de système de fichiers :** Si vous passez d'un projet utilisant **SPIFFS** à un projet utilisant **LittleFS**, il est préférable d'effacer toute la Flash pour éviter les conflits de partitions.
- **Après un crash sévère :** Si la mémoire a été corrompue suite à un bug de gestion de la mémoire, un effacement total remet la carte à "l'état d'usine".
- **Vente ou don de la carte :** Pour être sûr qu'aucune donnée sensible (mot de passe WiFi, clés API) ne reste stockée dans les recoins de la mémoire.

Les conséquences

Attention : Comme son nom l'indique, cette opération est irréversible.

- Toutes vos **préférences sauvegardées** (via la bibliothèque `Preferences.h`) seront supprimées.
- Tous vos **fichiers stockés** (images, pages web, logs) seront effacés.
- Le temps de téléversement sera un peu plus long car l'IDE doit envoyer une commande d'effacement complet (`chip_erase`) avant d'écrire.

Comment l'activer ?

1. Allez dans le menu **Outils** (Tools).
2. Cherchez la ligne **Erase All Flash Before Sketch Upload**.
3. Sélectionnez **Enabled** (Activé).
4. **Important :** Une fois votre problème résolu, remettez cette option sur **Disabled** pour gagner du temps lors de vos prochains essais.

Events Run On

L'option **Events Run On** est un réglage avancé, spécifique à l'architecture de l'**ESP32**, qui possède la particularité d'avoir **deux cœurs** (Core 0 et Core 1) au lieu d'un seul.

Cette option vous permet de choisir sur quel "cerveau" le système doit gérer les **événements système** (principalement le WiFi, le Bluetooth et les tâches réseau TCP/IP).

Pourquoi séparer les tâches ?

Par défaut, l'ESP32 répartit son travail ainsi :

- **Core 1 :** C'est là que tourne votre code Arduino principal (les fonctions `setup()` et `loop()`).
- **Core 0 :** C'est généralement là que le "système d'exploitation" (FreeRTOS) gère les communications sans fil complexes.

L'option **Events Run On** vous permet de forcer la gestion des événements sur l'un ou l'autre.

Les options disponibles :

1. **Core 0 (Par défaut/Recommandé) :** Le système gère le WiFi et le Bluetooth sur le cœur 0. Cela laisse le cœur 1 totalement libre pour exécuter votre code sans être interrompu. C'est la configuration la plus stable pour la majorité des projets.
2. **Core 1 :** Le système gère tout sur le même cœur que votre code. Cela peut être utile dans des cas très spécifiques de synchronisation, mais c'est risqué : si votre code fait un calcul trop long (comme une grosse boucle `delay()`), il peut empêcher le WiFi de répondre, provoquant une déconnexion.

Vous n'aurez probablement jamais besoin de changer ce réglage, sauf si :

- **Vous développez une application temps réel critique :** Si vous avez besoin d'une précision absolue sur le Core 1 (ex: contrôler un moteur très précisément), vous voulez absolument que les événements WiFi (Core 0) ne viennent pas perturber votre rythme.
- **Optimisation de la latence :** Dans des projets réseau très pointus, déplacer la gestion des événements peut parfois réduire de quelques microsecondes le temps de réponse.

Note importante : Si vous changez ce paramètre et que votre ESP32 commence à redémarrer sans cesse ou que le WiFi devient instable, revenez immédiatement au réglage par défaut (**Core 0**).

Flash Frequency

L'option **Flash Frequency** (Fréquence de la Flash) définit la vitesse de communication entre le processeur de l'ESP32 et sa puce de mémoire Flash externe.

Contrairement à un ordinateur où la mémoire vive (RAM) et le stockage sont très distincts, l'ESP32 va chercher les instructions de votre programme directement dans sa puce Flash pour les exécuter. La vitesse de cet échange influence donc directement la vitesse globale d'exécution de votre code.

Dans l'Arduino IDE, vous aurez généralement le choix entre plusieurs fréquences :

- **40 MHz :** C'est le réglage le plus sûr et le plus stable. C'est la vitesse standard qui fonctionne avec 100% des modules ESP32 du marché.
- **80 MHz :** C'est le réglage "performance". La communication est deux fois plus rapide. La plupart des modules modernes (ESP32-WROOM, etc.) supportent très bien cette fréquence.

Pourquoi choisir l'un ou l'autre ?

1. Gain de performance (80 MHz)

En passant à 80 MHz, votre programme se charge plus vite en mémoire. Si vous avez un code très lourd ou si vous manipulez beaucoup de fichiers stockés sur la Flash (images pour un écran, fichiers serveurs web), vous verrez une amélioration de la réactivité.

2. Stabilité et Fiabilité (40 MHz)

Si votre carte commence à avoir des comportements étranges :

- Le code plante de manière aléatoire.
- Le message d'erreur `Flash read err` apparaît dans le Moniteur Série.
- Le téléversement échoue régulièrement. **Alors, repassez à 40 MHz.** Certaines puces Flash de basse qualité ou certains câblages trop longs sur des modules "maison" ne supportent pas la haute vitesse.

La vitesse n'est qu'une partie de l'équation. Elle travaille souvent en tandem avec le **Flash Mode** (comme QIO ou DIO).

- **40 MHz + DIO :** Sécurité maximale.

- **80 MHz + QIO :** Performance maximale (vitesse doublée et 4 fils de données au lieu de 2).

Conseil pratique : Pour la plupart des projets, vous pouvez laisser ce réglage sur **80 MHz**. Si vous remarquez que votre ESP32 chauffe anormalement ou redémarre sans raison, essayez de descendre à **40 MHz** pour diagnostiquer le problème.

Flash Mode

Le **Flash Mode** définit la manière dont le processeur de l'ESP32 communique avec sa puce de mémoire Flash (où est stocké votre programme). C'est un réglage de "largeur de bande" : il détermine combien de fils de données sont utilisés simultanément pour lire les instructions.

C'est un peu comme comparer une route à une ou plusieurs voies : plus il y a de voies, plus les données circulent vite.

Dans le menu de l'Arduino IDE, vous trouverez généralement ces quatre options :

1. QIO (Quad I/O) - Le plus rapide

- **Fonctionnement :** Utilise **4 fils** de données pour les adresses et pour les données.
- **Performance :** C'est le mode le plus performant.
- **Compatibilité :** Nécessite que la puce Flash et le câblage supportent le mode Quad. Si votre ESP32 ne démarre plus après un téléversement dans ce mode, c'est que votre matériel n'est pas compatible.

2. QOUT (Quad Output)

- **Fonctionnement :** Utilise **4 fils** uniquement pour les données (les adresses passent par un seul fil).
- **Performance :** Un peu moins rapide que le QIO, mais plus stable sur certaines puces.

3. DIO (Dual I/O) - Le standard

- **Fonctionnement :** Utilise **2 fils** de données pour les adresses et les données.
- **Performance :** Vitesse moyenne.
- **Usage :** C'est le mode par défaut pour beaucoup de modules ESP32 car il est très fiable.

4. DOUT (Dual Output) - Le plus compatible

- **Fonctionnement :** Utilise **2 fils** uniquement pour les données.
- **Performance :** Le plus lent des quatre.
- **Usage :** À utiliser en dernier recours si aucun autre mode ne fonctionne. Il est compatible avec quasiment toutes les puces Flash du marché.

Le choix dépend du modèle exact de votre module ESP32 (le petit carré métallique sur votre carte) :

- **Pour un ESP32-WROOM ou WROVER classique :** Le mode **DIO** est le choix le plus sûr. Beaucoup supportent le **QIO**, mais le gain de vitesse n'est pas toujours perceptible pour des programmes simples.
- **Si vous avez besoin de performance (affichage graphique, audio) :** Essayez le mode **QIO**. Si la carte redémarre en boucle avec un message d'erreur "Flash read err", redescendez en **DIO**.
- **Attention aux broches utilisées :** Le mode Quad (QIO/QOUT) utilise des broches spécifiques (GPIO 6 à 11). Si vous essayez d'utiliser ces broches pour autre chose dans votre projet, le mode Quad créera un conflit et votre programme plantera.

En résumé :

Mode	Nombre de fils de données	Vitesse relative	Stabilité
QIO	4	⭐ ⭐ ⭐ ⭐	Moyenne
QOUT	4	⭐ ⭐ ⭐	Bonne
DIO	2	⭐ ⭐	Excellente
DOUT	2	⭐	Maximale

Flash Size

L'option **Flash Size** (Taille de la Flash) indique à l'Arduino IDE la capacité de stockage totale de la puce mémoire présente sur votre module ESP32.

Contrairement à un ordinateur où le système détecte automatiquement la taille du disque dur, ici, le compilateur a besoin de connaître cette limite exacte pour organiser l'espace (le "partitionnement") avant d'envoyer votre programme.

La mémoire Flash de l'ESP32 contient tout ce qui doit survivre à une coupure de courant :

1. **Votre programme** (le code compilé).
2. **Le système d'exploitation** (FreeRTOS).
3. **Les données utilisateur** (fichiers LittleFS/SPIFFS, réglages WiFi).
4. **La zone OTA** (Over-The-Air) : une zone réservée pour mettre à jour la carte sans fil.

Si vous réglez l'IDE sur **2MB** alors que votre carte possède **4MB**, vous n'utiliserez que la moitié de la capacité disponible. À l'inverse, si vous réglez sur **4MB** pour une carte qui n'en a que **2MB**, le téléversement échouera ou la carte plantera dès qu'elle cherchera à accéder à une zone mémoire inexistante.

Les tailles courantes pour l'ESP32

- **4MB (4 Mega Octets) :** C'est le standard absolu. La grande majorité des modules **ESP32-WROOM-32** vendus dans le commerce possèdent 4MB.
- **2MB :** Souvent présent sur des versions miniatures ou très low-cost.
- **8MB / 16MB :** Présent sur des cartes haut de gamme (comme certains modèles M5Stack ou des ESP32-S3) destinées à stocker beaucoup d'images ou de sons.

Comment connaître la taille de sa carte ?

Si vous ne connaissez pas la capacité de votre carte, vous pouvez utiliser l'option **"Get Board Info"** que nous avons vue précédemment, mais elle ne donne pas toujours la taille exacte de la Flash.

La méthode la plus fiable consiste à regarder le Moniteur Série au moment du démarrage ou lors du téléversement. L'outil `esptool` (utilisé par l'IDE) affiche souvent une ligne du type :

```
Detected Flash size: 4MB
```

La taille de la Flash est indissociable du **Partition Scheme** (Schéma de partition). Par exemple, sur une carte de **4MB**, vous pouvez choisir de répartir l'espace ainsi :

- **Default :** ~1.2MB pour le code, 1.5MB pour les fichiers, le reste pour les mises à jour.
- **No OTA (Large APP) :** ~3MB pour le code (si votre programme est énorme), mais vous perdez la possibilité de mettre à jour la carte via WiFi.

Conseil : Si vous avez un doute, laissez ce réglage sur **4MB**. C'est le réglage "universel" pour 90 % des ESP32 de développement (DevKit V1, NodeMCU-32S, etc.).

JTAG Adapter

L'option **JTAG Adapter** dans l'Arduino IDE pour ESP32 est un réglage de pointe qui permet de choisir l'outil matériel utilisé pour le **débogage en temps réel**.

Alors que le *Serial Monitor* ne permet que d'afficher du texte pour comprendre ce qui se passe, le **JTAG** permet de "figer" le processeur, d'inspecter chaque variable et de parcourir le code ligne par ligne pendant qu'il s'exécute.

Le JTAG (*Joint Test Action Group*) est une interface physique standard utilisée pour tester et déboguer les microcontrôleurs. Sur un ESP32, cela nécessite généralement un boîtier externe (l'adaptateur) branché entre votre ordinateur et des broches spécifiques de la carte.

À quoi sert cette option dans l'IDE ?

Dans Arduino IDE 2.x, vous avez un bouton "Start Debugging" (l'icône de l'insecte avec une flèche de lecture). Pour que ce bouton fonctionne, l'IDE doit savoir quel matériel fait le pont entre votre PC et la puce.

Voici les choix courants que vous trouverez dans le menu :

1. **Integrated JTAG (ESP32-S3 / C3) :** Les puces récentes comme l'ESP32-S3 intègrent leur propre convertisseur JTAG. Vous n'avez besoin d'aucun matériel supplémentaire, juste d'un câble USB.
2. **ESP-Prog :** C'est l'outil officiel d'Espressif. C'est un petit circuit imprimé que l'on relie aux broches GPIO de l'ESP32.
3. **J-Link / FT2232 :** Des adaptateurs professionnels universels très rapides.

Pourquoi l'utiliser ?

- **Arrêt sur image (Breakpoints) :** Vous pouvez dire à l'ESP32 : "Arrête-toi dès que tu arrives à la ligne 42". Vous pouvez alors vérifier si vos variables ont les bonnes valeurs.
- **Analyse des crashs :** Si votre carte redémarre sans cesse, le JTAG permet de voir exactement quelle instruction a provoqué l'erreur système.
- **Rapidité :** Pour les très gros projets, le JTAG permet de téléverser le code plus rapidement que par le port série classique.

Pourquoi ne pas toujours l'utiliser ?

- **Complexité :** Cela demande de configurer des pilotes spécifiques (souvent avec un utilitaire comme *Zadig*).
- **Broches occupées :** Le JTAG utilise plusieurs broches GPIO (souvent 12, 13, 14 et 15). Si vous utilisez ces broches pour des capteurs ou des écrans, vous ne pourrez pas utiliser le JTAG en même temps.

Conseil : Si vous débutez et que vous n'avez pas de boîtier "ESP-Prog" ou une puce de série "S", laissez cette option par défaut. Le débogage par `Serial.print()` reste la méthode la plus simple et la plus utilisée.

Arduino Runs On

L'option **Arduino Runs On** est très similaire à l'option *Events Run On* que nous avons vue précédemment, mais elle concerne cette fois-ci **votre code à vous** (le "Sketch").

Comme l'ESP32 possède deux cœurs (Core 0 et Core 1), cette option vous permet de décider quel cerveau sera chargé d'exécuter vos fonctions `setup()` et `loop()`.

Sur un ESP32 standard, vous aurez le choix entre :

1. **Core 1 (Par défaut / Recommandé) :** C'est le réglage standard. Votre code s'exécute sur le cœur 1, tandis que le cœur 0 est réservé aux tâches "nobles" du système (WiFi, Bluetooth, pile TCP/IP).
    - **Avantage :** Votre programme ne ralentit pas la connexion sans fil, et inversement.
2. **Core 0 :** Votre code est déplacé sur le même cœur que les fonctions réseau.
    - **Usage :** C'est très rare d'utiliser ce réglage. On le fait parfois pour libérer totalement le Core 1 afin d'y lancer manuellement des tâches ultra-prioritaires et chronométrées à la microseconde près via FreeRTOS (le système d'exploitation interne).

L'ESP32 utilise un système d'exploitation temps réel appelé **FreeRTOS**. Même si vous avez l'impression d'écrire un code simple, l'IDE crée en arrière-plan une "tâche" (Task) qui contient votre boucle `loop()`.

- **Si vous choisissez Core 1 :** Si votre code "plante" ou s'arrête (par exemple avec une boucle infinie mal codée), le WiFi sur le Core 0 peut continuer de fonctionner quelques instants avant que le chien de garde (*Watchdog*) ne s'aperçoive du problème.
- **Si vous choisissez Core 0 :** Si votre code est trop lourd ou contient des `delay()` trop longs, vous allez affamer le processeur. Le WiFi n'aura plus assez de temps de calcul pour maintenir la connexion et votre carte se déconnectera sans cesse.

En résumé :

Option	Destination de votre code	Utilisation
Core 1	Cœur Applicatif	**99% du temps.** C'est la configuration stable.
Core 0	Cœur Système	Projets très avancés où l'on veut inverser les rôles.

Partition Scheme

L'option **Partition Scheme** (Schéma de partition) est l'un des réglages les plus importants pour l'ESP32. Elle définit comment la mémoire Flash totale de votre carte (par exemple 4 Mo) est découpée en "tiroirs" virtuels.

Puisque l'ESP32 doit stocker à la fois votre code, les données du système, et parfois des fichiers (images, pages Web), il faut lui dire à l'avance quelle taille accorder à chaque section.

Par défaut, l'espace est divisé pour permettre les mises à jour sans fil (**OTA** - *Over The Air*). Cela signifie que l'IDE réserve deux zones identiques pour votre programme : une pour le code actuel et une vide pour recevoir la mise à jour.

Conséquence : Sur 4 Mo, vous ne pouvez utiliser qu'environ 1,2 Mo pour votre code réel. Si votre programme dépasse cette taille, vous recevrez une erreur de type *"Sketch too big"*. C'est là qu'intervient le changement de partition.

Les options les plus courantes

Voici les réglages que vous trouverez généralement dans le menu **Outils > Partition Scheme** :

- **Default (4MB with OTA) :**
    - Code : ~1,2 Mo
    - Fichiers (SPIFFS/LittleFS) : ~1,5 Mo
    - *Usage :* Le standard pour les projets classiques avec mise à jour WiFi.

- **No OTA (2MB APP / 2MB SPIFFS) :**
  - Code : 2 Mo
  - Fichiers : 2 Mo
  - *Usage :* Pour les projets avec beaucoup d'images ou de données, mais sans mise à jour sans fil.
- **Huge APP (3MB No OTA) :**
  - Code : **3 Mo**
  - Fichiers : Très peu (environ 190 Ko).
  - *Usage :* Indispensable pour les programmes énormes (intelligence artificielle simple, Bluetooth lourd, ou interfaces graphiques complexes).
- **Minimal SPIFFS :**
  - Donne presque tout l'espace au code et réduit la zone de fichiers au strict minimum.

Quel est l'impact sur votre code ?

Le choix de la partition modifie la structure de la mémoire **pendant le téléversement**.

Attention : Si vous changez de schéma de partition, l'emplacement des données change. Cela peut effacer les fichiers que vous aviez stockés dans la mémoire SPIFFS ou LittleFS. Il est donc conseillé de choisir son schéma au début du projet et de ne plus en changer.

Si vous utilisez ESP32forth avec beaucoup d'options, privilégiez la seconde option :

Tools → Partition Scheme :

- "No OTA (2 MB APP / 2 MB SPIFFS)" = 2 MB pour votre code
- "**Huge APP (3 MB No OTA)**" = 3 MB pour votre code ESP32forth

Même si ça réduit l'espace mémoire FLASH pour SPIFFS, ça laisse suffisamment de place pour les codes sources FORTH.

Comment choisir ?

1. **Votre code est trop gros ?** Passez en **"Huge APP"**.
2. **Vous avez besoin de stocker beaucoup de fichiers (Sons, HTML) ?** Passez en **"No OTA (Large SPIFFS)"**.
3. **Vous voulez pouvoir mettre à jour votre carte par WiFi ?** Restez sur un mode **"with OTA"**.

Consulter le rapport de compilation

Une bonne habitude à prendre : surveiller la taille de son programme permet d'éviter bien des frustrations au moment du téléversement.

Voici comment lire et interpréter ces informations dans l'**Arduino IDE**.

Chaque fois que vous cliquez sur **Vérifier** (l'icône coche) ou **Téléverser**, l'IDE affiche un résumé dans la console noire en bas de la fenêtre.

Voici à quoi cela ressemble :

```
The sketch uses 262,144 bytes (20%) of the program storage
space. The maximum is 1,310,720 bytes. Global variables use
15,320 bytes (4%) of the dynamic memory...
```

- **Espace de stockage (Flash) :** C'est la taille de votre "tiroir" de code définie par votre **Partition Scheme**. Si vous atteignez 90-95%, il est temps de passer à une partition plus grande (comme *Huge APP*).

- **Mémoire dynamique (RAM) :** C'est la mémoire vive. Si elle est trop haute (proche de 100%), votre ESP32 risque de "freezer" ou de redémarrer de manière imprévisible pendant l'exécution.

PSRAM

La **PSRAM** (Pseudo-Static Random Access Memory) est une extension de la mémoire vive (RAM) de l'ESP32.

Pour comprendre son utilité, imaginez que la RAM interne de l'ESP32 est un petit bureau très rapide pour travailler. Si vous avez un projet énorme (traitement d'images, audio, intelligence artificielle), ce bureau devient trop petit. La PSRAM est comme une **grande table d'architecte ajoutée à côté** : elle est un peu moins rapide que le bureau principal, mais elle offre énormément d'espace supplémentaire.

L'ESP32 classique possède environ **520 Ko** de RAM interne. C'est beaucoup pour de l'électronique simple, mais très peu pour :

- **La Vidéo/Photo :** Stocker une seule image haute résolution provenant d'une caméra (ESP32-CAM).
- **L'Audio :** Créer un tampon (buffer) pour lire de la musique en streaming sans coupure.
- **L'Affichage :** Gérer des écrans tactiles couleurs avec des interfaces graphiques fluides (LVGL).
- **Le Web :** Faire tourner un serveur complexe avec beaucoup de données dynamiques.

Avec la PSRAM, vous pouvez passer de 520 Ko à **2 Mo, 4 Mo ou même 8 Mo** de mémoire disponible.

Si votre carte possède une puce de PSRAM (comme les modèles **WROVER** ou **ESP32-S3-WROOM-1**), elle n'est pas forcément active par défaut.

1. Allez dans le menu **Outils**.
2. Cherchez l'option **PSRAM**.
3. Sélectionnez **Enabled** (ou "OPI PSRAM" / "QSPI PSRAM" selon votre modèle).

Les limites à connaître

- **Vitesse :** La PSRAM est branchée via un bus série (SPI). Elle est donc plus lente que la RAM interne. On y stocke les grosses données, mais pas les calculs mathématiques ultra-rapides.
- **Consommation :** Elle consomme un peu plus d'énergie, ce qui est à surveiller sur batterie.
- **Conflit de broches :** Sur certains vieux modèles d'ESP32, l'utilisation de la PSRAM "vole" certaines broches GPIO (souvent 16 et 17). Vérifiez bien votre schéma (pinout).

Comment savoir si vous en avez ? Regardez l'étiquette sur le module métallique de votre carte. Si vous voyez écrit **WROVER**, vous avez de la PSRAM. Si c'est écrit **WROOM**, vous n'en avez généralement pas (sauf versions spéciales).

Upload Speed

L'option **Upload Speed** (Vitesse de téléversement) définit la vitesse à laquelle l'Arduino IDE envoie votre programme compilé vers la carte ESP32 via le câble USB.

Cette vitesse est mesurée en **bauds** (bits par seconde). C'est le "débit" du tuyau entre votre ordinateur et votre carte pendant la phase de programmation.

Contrairement aux cartes Arduino Uno qui sont limitées à 115 200 bauds, l'ESP32 est un véritable bolide. Vous trouverez généralement ces options :

- **115 200 :** La vitesse "de sécurité". C'est lent mais très fiable.
- **460 800 :** Un excellent compromis, très utilisé.
- **921 600 :** La vitesse maximale courante. C'est presque 10 fois plus rapide que l'Arduino standard.

Pourquoi choisir une vitesse plutôt qu'une autre ?

1. Le gain de temps

Si vous développez un gros projet (avec beaucoup de bibliothèques comme le WiFi ou le Bluetooth), votre fichier binaire peut peser plusieurs mégaoctets.

- À **115 200**, le téléversement peut prendre 30 à 40 secondes.
- À **921 600**, cela ne prend que 5 à 10 secondes. Sur une journée de travail avec des dizaines de tests, la différence est énorme.

2. La fiabilité (Stabilité)

Si vous réglez la vitesse trop haut, vous risquez des erreurs de transfert (`Packet content mismatch` ou `Failed to connect`). Cela arrive généralement à cause de :

- **Un câble USB de mauvaise qualité :** Trop long ou mal blindé.
- **Un convertisseur USB-Série bon marché :** Certaines cartes "clones" utilisent des puces qui ne supportent pas bien les hautes vitesses.
- **Des interférences :** Si votre montage comporte des moteurs ou des alimentations bruyantes à côté.

Est-ce que cela change la vitesse de mon programme ?

Non. C'est une confusion fréquente.

- **Upload Speed :** Ne concerne que le transfert du code (la phase de programmation).
- **Baud Rate (Serial.begin) :** C'est la vitesse de communication pendant que le programme tourne (pour le Moniteur Série).

Vous pouvez très bien téléverser à **921 600** et communiquer avec le Moniteur Série à **115 200**

Zigbee Mode

L'option **Zigbee Mode** est apparue avec les nouvelles générations de puces, spécifiquement l'**ESP32-H2** et l'**ESP32-C6**. Contrairement à l'ESP32 classique qui ne gère que le WiFi et le Bluetooth, ces puces intègrent une radio compatible avec la norme **IEEE 802.15.4**.

Cette option dans l'Arduino IDE permet de définir le rôle que votre carte va jouer au sein d'un réseau domestique intelligent (domotique).

Le Zigbee est l'alternative idéale au WiFi pour les objets connectés (IoT) car :

- **Consommation ultra-faible :** Un capteur peut fonctionner des années sur une pile bouton.
- **Réseau Maillé (Mesh) :** Chaque appareil branché sur secteur peut répéter le signal pour étendre la portée.
- **Standard domotique :** C'est le langage utilisé par Philips Hue, IKEA TRÅDFRI ou les capteurs Xiaomi.

Les différents modes disponibles

Dans le menu, vous aurez généralement le choix entre trois types d'appareils (Device Types) :

1. Zigbee Coordinator (Coordonnateur)

C'est le "cerveau" du réseau.

- **Rôle :** Il crée le réseau, autorise les nouveaux appareils à se connecter et gère la sécurité.
- **Note :** Il ne peut y avoir qu'un seul coordonnateur par réseau Zigbee (souvent votre box domotique ou une clé USB dédiée).

2. Zigbee Router (Routeur)

C'est un appareil intermédiaire qui reste toujours alimenté (ex: une ampoule ou une prise connectée).

- **Rôle :** Il exécute ses tâches (allumer la lumière) tout en relayant les messages des autres capteurs vers le coordonnateur.

3. Zigbee End Device (Appareil final)

C'est l'appareil qui dort la plupart du temps pour économiser sa batterie (ex: un capteur de mouvement ou de température).

- **Rôle :** Il se réveille, envoie sa donnée, et se rendort. Il ne relaie jamais les messages des autres.

Pourquoi ce réglage est-il dans l'IDE ?

L'ESP32 doit configurer sa pile logicielle (stack) différemment selon le mode choisi. Par exemple, si vous choisissez **End Device**, l'IDE inclura des fonctions de gestion de sommeil profond (*Deep Sleep*). Si vous choisissez **Router**, il activera les fonctions de routage réseau.

Matter et Zigbee

Notez que ces puces sont aussi la base du nouveau standard **Matter**. Le mode Zigbee est souvent la première étape pour créer des ponts (Bridges) entre vos anciens appareils Zigbee et vos nouveaux appareils Matter.

Attention : Si vous possédez un ESP32 classique (WROOM-32), cette option n'apparaîtra pas ou n'aura aucun effet, car le matériel radio n'est pas compatible.

Les cartes ESP32

L'écosystème de l'ESP32 s'est énormément diversifié. Aujourd'hui, on ne parle plus d'une seule "carte", mais d'une vaste famille de puces adaptées à des besoins précis (puissance, coût ou connectivité).

Avant de choisir une carte de développement (le PCB), il faut choisir la puce qui est dessus.

Série	Usage principal	Caractéristiques clés
ESP32 (Original)	Polyvalence	Dual-core, Wi-Fi, Bluetooth Classic + BLE. C'est la base, très robuste.
ESP32-S2	Sécurité & USB	Mono-cœur, **USB natif**, plus de broches GPIO, mais **pas de Bluetooth**.
ESP32-S3	**IA & Vidéo**	Dual-core, USB natif, instructions pour l'IA, idéal pour la reconnaissance vocale ou d'image.
ESP32-C3 / C6	Bas coût / IoT	Mono-cœur (RISC-V). Le **C6** supporte le **Wi-Fi 6** et **Matter**.
ESP32-H2	Domotique Mesh	**Pas de Wi-Fi**. Uniquement Bluetooth et Zigbee/Thread.

Les formats de cartes les plus courants

Une fois la puce choisie, elle est montée sur une carte de développement prête à l'emploi. Les classiques (Breadboard friendly) :

- **ESP32 DevKit V1 (NodeMCU) :** La plus célèbre. Elle possède 30 ou 38 broches, un port micro-USB et deux boutons (Boot et EN). Parfaite pour débuter.
- **ESP32-WROOM / WROVER :** Ce sont souvent des dénominations de modules. Le *WROVER* possède généralement de la mémoire supplémentaire (PSRAM), utile pour le traitement d'images.

Les cartes ESP32 spécialisées :

- **ESP32-CAM :** Inclut un petit capteur caméra (OV2640) et un lecteur de carte SD. Très populaire pour la surveillance.
- **M5Stack :** Des cartes enfermées dans un petit boîtier avec écran, batterie et connecteurs Grove. C'est le "LEGO" de l'ESP32.
- **TinyPICO / LOLIN D32 :** Des formats miniatures pour les projets où l'espace est compté.

Les cartes ESP32 modernes (USB-C & Compact) :

- **XIAO ESP32 (Seeed Studio) :** Minuscule, format timbre-poste, souvent en USB-C.
- **LilyGO T-Display :** Intègre directement un écran LCD couleur, idéal pour faire des interfaces sans câblage complexe.

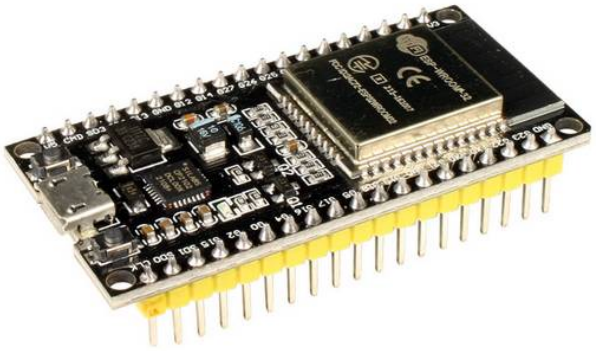
Laquelle choisir pour votre projet :

- **"Je débute totalement"** Prenez une **ESP32 DevKit V1** classique. Elle est peu chère (environ 5-8€) et tous les tutoriels sur internet l'utilisent.
- **"Je veux faire de la domotique moderne"** Optez pour l'**ESP32-C6**. Sa compatibilité avec la norme **Matter** et le Wi-Fi 6 en fait le meilleur choix pour l'avenir.

- **"J'ai besoin de reconnaissance faciale ou d'audio"** Dirigez-vous vers l'**ESP32-S3**. Sa puissance de calcul et ses instructions vectorielles sont faites pour ça.

ESP32 WROOM

La carte **ESP32-WROOM** (souvent appelée DevKit V1 dans sa version complète) est la référence absolue pour les makers. Elle est basée sur le module WROOM-32 qui intègre la puce originale d'Espressif.



Voici ses caractéristiques techniques détaillées :

Le Cœur du Système (Microcontrôleur)

C'est là que réside toute la puissance de la carte. Contrairement à un Arduino classique, nous sommes sur une architecture beaucoup plus musclée.

- **Processeur :** Xtensa® Dual-Core 32-bit LX6.
- **Fréquence d'horloge :** Ajustable de 80 MHz à **240 MHz**.
- **Mémoire interne :** 520 Ko de SRAM.
- **Mémoire Flash :** Généralement **4 Mo** (pour stocker votre code et vos fichiers).

Connectivité Sans Fil

C'est le point fort de l'ESP32. Elle est nativement "connectée".

- **Wi-Fi :** 802.11 b/g/n (jusqu'à 150 Mbps). Peut fonctionner en mode Station (se connecte à votre box) ou en Point d'Accès (crée son propre réseau).
- **Bluetooth :** Double mode, supportant le **Bluetooth Classic** (audio, transfert de données) et le **Bluetooth Low Energy (BLE)** (basse consommation, idéal pour les capteurs).

Entrées et Sorties (GPIO)

La carte DevKit possède généralement **30 ou 38 broches**. Elle offre une grande flexibilité pour les capteurs et actionneurs :

- **ADC (Convertisseurs Analogique-Numérique) :** Jusqu'à 18 canaux en 12 bits (très précis).
- **DAC (Convertisseurs Numérique-Analogique) :** 2 canaux en 8 bits pour générer de vrais signaux analogiques.
- **Capteurs de toucher (Touch Sensors) :** 10 broches capacitives (permettent de créer des boutons tactiles sans composants externes).
- **Interfaces de communication :** 3x UART, 3x SPI, 2x I2C, CAN bus 2.0, et I2S (pour l'audio).
- **PWM :** 16 canaux indépendants (pour piloter des LEDs ou des servos).

Consommation d'Énergie

L'ESP32-WROOM est conçue pour les projets sur batterie grâce à ses différents modes de veille :

Mode	Consommation (approx.)	État
Actif	160 ~ 260 mA	Wi-Fi et CPU en marche.

Mode	Consommation (approx.)	État
Modem-sleep	20 ~ 68 mA	CPU actif, Wi-Fi coupé.
Light-sleep	0.8 mA	CPU en pause, réveil rapide.
Deep-sleep	**10 µA**	Presque tout est coupé sauf l'horloge (RTC).

Alimentation

- **Via USB :** 5V via le port micro-USB.
- **Via broche Vin :** Vous pouvez injecter du 5V régulé.
- **Tension logique : Attention, l'ESP32 fonctionne en 3.3V.** Envoyer du 5V sur une broche de données (GPIO) peut détruire la puce.

Note : Le module WROOM possède une antenne PCB intégrée (le tracé en zigzag sur le circuit), ce qui offre une portée Wi-Fi très correcte sans antenne externe.

Paramètres tools pour ESP32-WROOM

Voici les paramètres essentiels pour paramétrer l'option *tools* de l'IDE Arduino avant de compiler et téléverser ESP32forth :

```
Board: ESP32 Dev Module
Port: COMx
--------------------------------
CPU Frequency: 240 Mhz
Core Debug Level: None
Erase All Flash Before Sketch Upload: Disabled
Events Run On: Core 1
Flash Frequency: 80 Mhz
Flash Mode: QIO
Flash Size: 4MB
JTAG Adapter: Disabled
Arduino Runs On:
Partition Scheme: No OTA (2 MB APP / 2 MB SPIFFS)
PSRAM: Disabled
Upload Speed: 921600
ZigBee Mode: Disabled
```

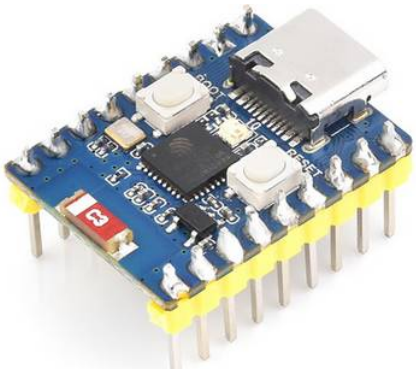
Paramètres tools pour ESP32-WROVER

Voici les paramètres essentiels pour paramétrer l'option *tools* de l'IDE Arduino avant de compiler et téléverser ESP32forth :

```
Board: ESP32 Dev Module
Port: COMx
--------------------------------
CPU Frequency: 240 Mhz
Core Debug Level: None
Erase All Flash Before Sketch Upload: Disabled
Events Run On: Core 1
Flash Frequency: 80 Mhz
Flash Mode: QIO
Flash Size: 4MB
JTAG Adapter: Disabled
Arduino Runs On:
Partition Scheme: No OTA (2 MB APP / 2 MB SPIFFS)
PSRAM: Enabled
Upload Speed: 921600
```

```
ZigBee Mode: Disabled
```

ESP3 C3 Zero



L'**ESP32-C3 Zero** est une carte de développement ultra-compacte (souvent produite par Waveshare ou des fabricants similaires) conçue pour être une alternative moderne et économique à l'ESP8266.

Contrairement à l'ESP32-S3 que vous évoquiez, le C3 utilise une architecture **RISC-V** monocœur. C'est le choix idéal pour les petits objets connectés qui n'ont pas besoin d'une puissance de calcul énorme mais nécessitent une faible consommation.

Caractéristiques Techniques (Hardware)

- **Processeur :** RISC-V 32 bits simple cœur, cadencé jusqu'à **160 MHz**.
- **Mémoire vive (SRAM) :** 400 Ko internes.
- **Stockage (Flash) :** 4 Mo intégrés.
- **Connectivité :**
  - Wi-Fi 4 (802.11 b/g/n) à 2,4 GHz.
  - Bluetooth 5.0 et **BLE (Bluetooth Low Energy)**.
- **Consommation :** Très faible en mode *Deep Sleep* (environ **43 µA**).
- **Dimensions :** Environ 23,5 × 18 mm (format timbre-poste).

Interfaces et Broches (I/O)

Bien que la carte soit minuscule, elle expose l'essentiel :

- **15 GPIOs** multifonctions.
- **6 entrées analogiques (ADC)** sur 12 bits.
- **Interfaces série :** 2x UART, 1x I2C, 3x SPI, 1x I2S.
- **USB natif :** Port USB-C avec contrôleur **CDC/JTAG intégré** (pas besoin de puce convertisseuse comme le CH340).
- **LED RGB :** Une LED WS2812 (Neopixel) est souvent intégrée sur la carte (généralement sur le **GPIO 8**).

Réglages conseillés dans l'IDE Arduino

Pour cette carte spécifique, voici comment ajuster vos paramètres :

- **Board :** "ESP32C3 Dev Module" (ou "Waveshare ESP32-C3-Zero" si disponible).
- **USB CDC On Boot : Enabled** (indispensable pour voir le port série via l'USB-C).
- **Flash Mode : DIO** (le mode QIO n'est pas toujours supporté par les puces intégrées sur ce petit format).
- **Partition Scheme :** "4MB with SPIFFS" ou "Minimal SPIFFS" selon vos besoins.
- **PSRAM : Disabled** (l'ESP32-C3 ne possède pas de PSRAM et ne peut pas en gérer de manière externe facilement).

Points de vigilance

1. **Absence de Bluetooth Classic :** Contrairement aux anciens ESP32, le C3 ne supporte **que le Bluetooth Low Energy (BLE)**. Vous ne pourrez pas l'utiliser comme une enceinte Bluetooth ou un clavier HID standard sans passer par le protocole BLE.
2. **Mode Bootloader :** Si vous n'arrivez plus à téléverser, maintenez le bouton **BOOT** (souvent marqué GPIO 9), appuyez brièvement sur **RESET**, puis relâchez BOOT. La carte apparaîtra alors comme un port série prêt à être flashé.

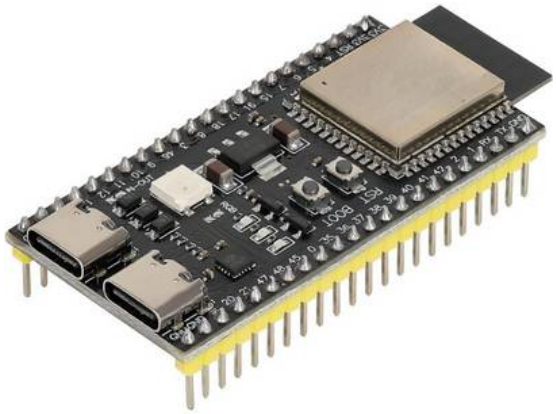
Paramètres tools pour ESP32 C3 Zero

Voici les paramètres essentiels pour paramétrer l'option *tools* de l'IDE Arduino avant de compiler et téléverser ESP32forth : @TODO à corriger

```
Board: ESP32C3 Dev Module
Port: COMx
--------------------------------
USB CDC On Boot: Enabled
CPU Frequency: 160 Mhz
Core Debug Level: None or Info
Erase All Flash Before Sketch Upload: Disabled
Flash Frequency: 80 Mhz
Flash Mode: QIO 80 Mhz
Flash Size: 4MB
JTAG Adapter: Disabled
Arduino Runs On:
Partition Scheme: No OTA (2 MB APP / 2 MB SPIFFS)
PSRAM: OPI PSRAM
Upload Speed: 921600
ZigBee Mode: Disabled
```

ESP32 S3 WROOM

L'**ESP32-S3-WROOM** est l'un des modules les plus puissants de la famille Espressif. C'est le successeur "boosté" du classique ESP32-WROOM, particulièrement optimisé pour l'**Intelligence Artificielle (AIoT)** et le traitement du signal.



Voici l'essentiel à savoir :

Puissance de calcul et IA

Contrairement à l'ESP32 classique (cœur LX6), le S3 utilise l'architecture **Xtensa® LX7**.

- **Processeur :** Dual-core 32 bits, jusqu'à **240 MHz**.
- **Accélération IA :** Il possède des instructions spécifiques pour accélérer les calculs de réseaux de neurones (reconnaissance vocale, détection de visage).
- **Mémoire :** Il dispose de **512 Ko de SRAM** interne, mais il est souvent accompagné de **PSRAM** (mémoire vive externe jusqu'à 16 Mo) pour gérer des projets lourds (écrans haute résolution, audio, etc.).

Connectivité

- **Wi-Fi :** 802.11 b/g/n (2.4 GHz).

- **Bluetooth :** Version **5.0 LE (Low Energy)**. *Attention : contrairement à l'ESP32 original, il ne gère plus le Bluetooth "Classic", seulement le LE.*
- **USB Natif :** C'est l'un de ses plus gros points forts. Il possède un contrôleur **USB OTG**, ce qui signifie qu'il peut être reconnu nativement comme un clavier, une souris ou une clé USB par un PC sans puce de conversion supplémentaire.

Les différentes versions

Le nom complet ressemble souvent à *ESP32-S3-WROOM-1-N16R8*. Voici comment décoder :

- **N4, N8, N16 :** Taille de la mémoire Flash (4, 8 ou 16 Mo).
- **R2, R8 :** Taille de la PSRAM (2 ou 8 Mo). S'il n'y a pas de "R", il n'y a pas de RAM supplémentaire.
- **WROOM-1 :** Antenne PCB intégrée.
- **WROOM-1U :** Connecteur pour antenne externe (U.FL).

Comparatif :

Caractéristique	ESP32 (Original)	ESP32-S3
Cœurs	LX6	**LX7 (plus récent)**
Instructions IA	Non	**Oui**
Bluetooth	4.2 + Classic	**5.0 LE + Mesh**
USB Natif	Non	**Oui (OTG)**
GPIO	~34	**~36 (plus flexibles)**

Si vous envisagez la reconnaissance vocale, piloter un écran complexe, ou besoin d'un port USB natif pour un projet, c'est la carte idéale.

Paramètres tools pour ESP32-S3-WROOM

Voici les paramètres essentiels pour paramétrer l'option *tools* de l'IDE Arduino avant de compiler et téléverser ESP32forth :

```
Board: ESP32S3 Dev Module
Port: COMx
--------------------------------
USB CDC On Boot: Enabled
CPU Frequency: 240 Mhz
Core Debug Level: None
Erase All Flash Before Sketch Upload: Disabled
Events Run On: Core 1
Flash Frequency: 80 Mhz
Flash Mode: QIO 80 Mhz
Flash Size: 4MB
JTAG Adapter: Disabled
Arduino Runs On:
Partition Scheme: No OTA (2 MB APP / 2 MB SPIFFS)
PSRAM: OPI PSRAM
Upload Speed: 921600
ZigBee Mode: Disabled
```

USB CDC On Boot

L'option **USB CDC On Boot** est spécifique aux cartes basées sur l'ESP32 (comme l'ESP32-S2, S3 ou C3) qui possèdent un support USB natif. En gros, cela détermine comment votre ordinateur "voit" la carte dès qu'elle est branchée.

CDC signifie *Communication Device Class*. C'est le protocole standard qui permet à un appareil USB d'être reconnu comme un **port série virtuel (COM)**. C'est ce qui permet d'utiliser le `Serial Monitor` pour déboguer votre code.

2. Les deux réglages possibles

- **Disabled (Désactivé) :** La carte n'active pas automatiquement le port série USB au démarrage. Si vous voulez afficher des messages dans le moniteur série, vous devez généralement passer par un composant matériel externe (convertisseur USB-UART) ou configurer manuellement l'USB dans votre code.
- **Enabled (Activé) :** C'est le réglage le plus courant pour les débutants. Dès que la carte s'allume, elle "dit" à l'ordinateur : *"Bonjour, je suis un port COM"*. Cela permet d'utiliser `Serial.print()` immédiatement sans configuration complexe.

Caractéristique	Enabled (Activé)	Disabled (Désactivé)
Usage principal	Débogage facile, envoi de données au PC.	Économie d'énergie, projets où l'USB n'est pas utilisé.
Port Série	Serial utilise le port USB natif.	Serial utilise souvent les broches matérielles (TX/RX).
Reconnexion	Le port COM apparaît dès le branchement.	Le port COM peut ne pas apparaître du tout.

Si vous téléversez un code avec **USB CDC On Boot: Disabled**, le port série risque de disparaître de votre gestionnaire de périphériques après le flashage.

PSRAM

La plupart des modules ESP32-S3 du commerce (comme ceux de Freenove, Lilygo ou les clones Amazon) possèdent de la mémoire vive externe (**PSRAM**).

- **Si votre carte a de la PSRAM :** Vous **devez** l'activer (choisissez généralement *OPI* ou *QSPI* selon votre modèle). Si vous laissez sur `Disabled`, vous ne pourrez pas utiliser cette mémoire, ce qui est dommage pour le traitement d'images ou l'affichage.
- **Conséquence d'un mauvais réglage :** Si vous activez "OPI" alors que votre carte n'en a pas (ou vice-versa), l'ESP32-S3 peut entrer dans une boucle de redémarrage infinie (*Bootloop*).

Ressources

- Site officiel ARDUINO IDE
  https://www.arduino.cc/en/software/

Forth

- Site de Brad NELSON, le développeur de ESP32forth, en anglais
  https://esp32forth.appspot.com/ESP32forth.html
- Site de Marc PETREMANN, en français et anglais
  https://esp32.arduino-forth.com/
- Site Forth Standard, en anglais
  https://forth-standard.org/