# ESP32

## Introduction aux microcontrôleurs modernes

Houssem-eddine LAHMER

# Les limites de l'Arduino classique

- Popularité historique (depuis 2005) mais technologie vieillissante
- Limitations techniques :
  - Fréquence d'horloge à 16 MHz
  - Connectivité sans fil nécessite des modules externes
  - Capacités de traitement limitées
  - Mémoire RAM restreinte (2-8 KB)
- Écosystème principalement orienté débutants

## Un besoin émergent

Nécessité croissante de :

- Connectivité native (WiFi/Bluetooth)
- Plus grande puissance de calcul
- Gestion d'énergie avancée
- Interfaces matérielles spécialisées

# L'ESP32 : Un successeur moderne

- Système sur puce (SoC) d'Espressif Systems
- Double coeur Xtensa LX6 (240 MHz)
- Connectivité intégrée :
  - WiFi 802.11 b/g/n
  - Bluetooth 4.2 + BLE
- Ultra basse consommation (10 uA en veille)
- 520 KB SRAM, 4-16 MB Flash
- 34 GPIOs programmables


Figure: Module ESP32 typique

# Évolution technologique

| **Caractéristique** | **ESP8266** | **ESP32** |
|---|---|---|
| Cores | Single | Dual |
| Fréquence | 80 MHz | 160/240 MHz |
| RAM | 80 KB | 520 KB |
| Bluetooth | Non | Oui |
| ADC | 10-bit | 12-bit |
| GPIO | 17 | 34 |
| Prix | \$1-\$2 | \$2-\$5 |

## Avantages clés

Rapport performance/prix exceptionnel + Connectivité native

# Qu'est-ce que l'ESP32 ?

## Définition

Microcontrôleur 32 bits **low-cost** avec :

- Architecture Xtensa® Dual-Core (240 MHz)
- Mémoire Flash: 4MB (extensible)
- RAM: 520KB SRAM + 8KB RTC
- Consommation: 0.15uA en veille profonde

- Intègre nativement :
  - WiFi 802.11 b/g/n (2.4 GHz)
  - Bluetooth 4.2 + BLE
  - 34 GPIOs multifonctions
  - 18 canaux ADC 12-bit
- Applications typiques : IoT, wearables, domotique, industrie 4.0

# Utilité de l'ESP32

- Alternative puissante aux cartes Arduino :
  - 240 MHz vs 16 MHz (Arduino Uno)
  - 520KB RAM vs 2KB
  - Connectivité intégrée vs modules externes
- Compatibilité Arduino IDE :
  - Même structure de code (setup/loop)
  - Librairies réutilisables
  - Workflow identique

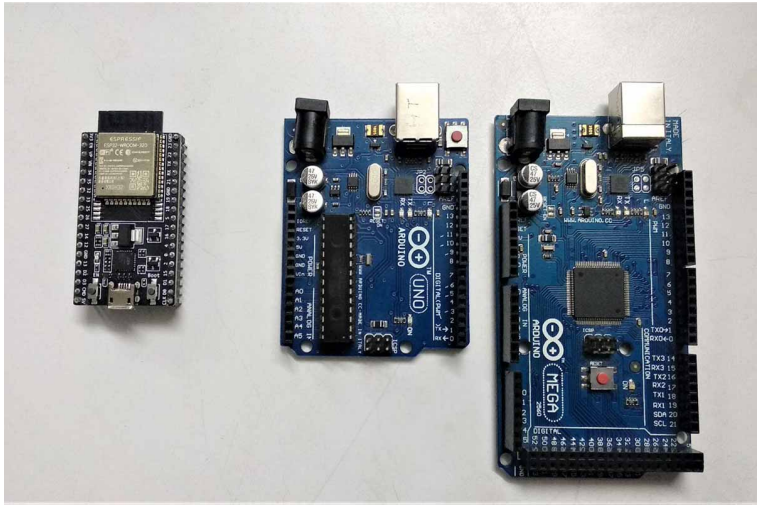

Figure: Comparaison de taille Arduino vs ESP32

**Avantage clé**

Miniaturisation extrême (3x5cm) + Écosystème mature + Coût ¡ 5$

# Arduino IDE pour ESP32


Figure: Interface Arduino IDE

- Environnement de développement familier
- Support natif via Boards Manager
- Mêmes concepts de base :
  - Structure setup()/loop()
  - Gestion des pins numérique/analogique
  - Bibliothèques compatibles
- Extensions spécifiques :
  - Gestion du dual core
  - Bluetooth/BLE
  - Deep sleep

# Connectivité avancée

## Sans fil

- WiFi :
  - Mode Station (client)
  - Mode Point d'accès
  - Débit jusqu'à 150Mbps
- Bluetooth :
  - Classique (audio, données)
  - BLE (capteurs IoT)
  - Mesh networking

## Interfaces matérielles

| | |
|---|---|
| GPIOs | 34 (dont 18 ADC) |
| DAC | 2 canaux 8-bit |
| Touch | 10 capteurs |
| PWM | 16 canaux |
| UART | 3 ports |
| SPI | 4 interfaces |
| I2C | 2 bus |

# Brochage de l'ESP32 DEVKIT

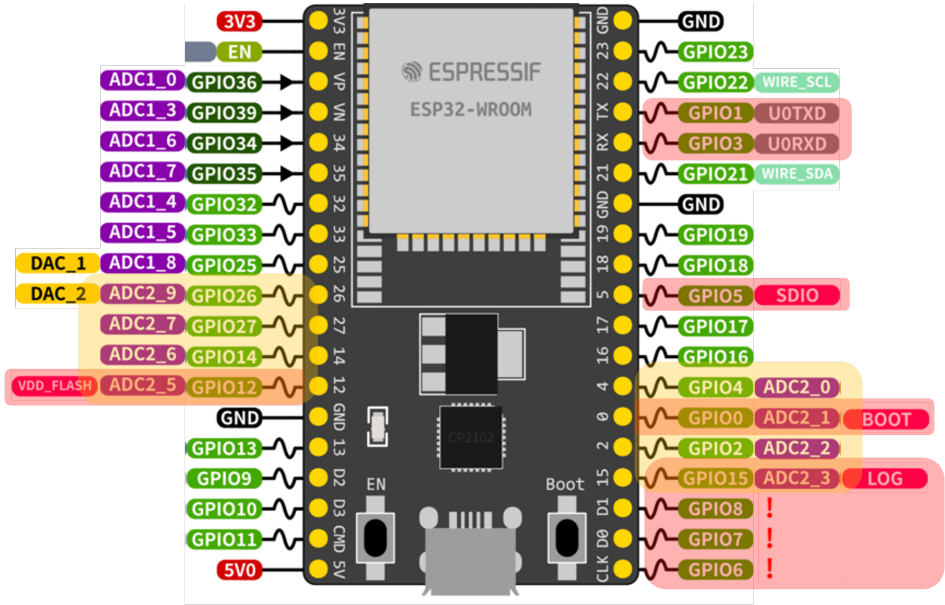


Figure: Détail des broches GPIO (Source: Espressif Systems)

- Broches multifonctions configurables :
  - Communication série (UART, SPI, I2C)
  - Entrées analogiques (ADC)
  - Sorties PWM
  - Capteurs capacitifs
- Alimentation flexible : 3.3V (USB) ou 5V (externe)

# ESP32 vs ESP8266 - Spécifications techniques

| Caractéristique | ESP8266 | ESP32 |
|---|---|---|
| **Architecture** | | |
| MCU | Xtensa Single-core 32-bit L106 | Xtensa Dual-Core 32-bit LX6 (600 DMIPS) |
| Fréquence | 80 MHz | 160 MHz |
| **Connectivité** | | |
| Wi-Fi | 802.11 b/g/n (HT20) | 802.11 b/g/n (HT40) |
| Bluetooth | Non | 4.2 + BLE |
| **Mémoire** | | |
| SRAM | 80 KB | 520 KB |
| Flash | 1-16 MB | 4-16 MB |
| **Entrées/Sorties** | | |
| GPIO | 17 | 36 |
| ADC | 10-bit | 12-bit |
| PWM | Logiciel (8 canaux) | Logiciel (16 canaux) |
| **Capteurs** | | |
| Touch Sensor | Non | Oui |
| Capteur Hall | Non | Oui |
| Température | Non | Oui |

## Avantages clés ESP32

Dual-core • Bluetooth • Meilleures performances • Plus d'I/O • Capteurs intégrés

# Comparaison visuelle

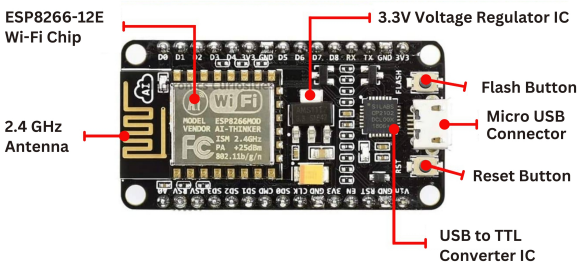


Figure: ESP8266 (Modèle ESP-12F)

Figure: ESP32 DevKit v1

- Différences physiques notables :
  - Taille des modules (ESP32 généralement plus grand)
  - Nombre de broches GPIO visibles
  - Composants supplémentaires sur l'ESP32 (antenne PCB, régulateurs)

# Layout typique d'une carte ESP32 DevKit

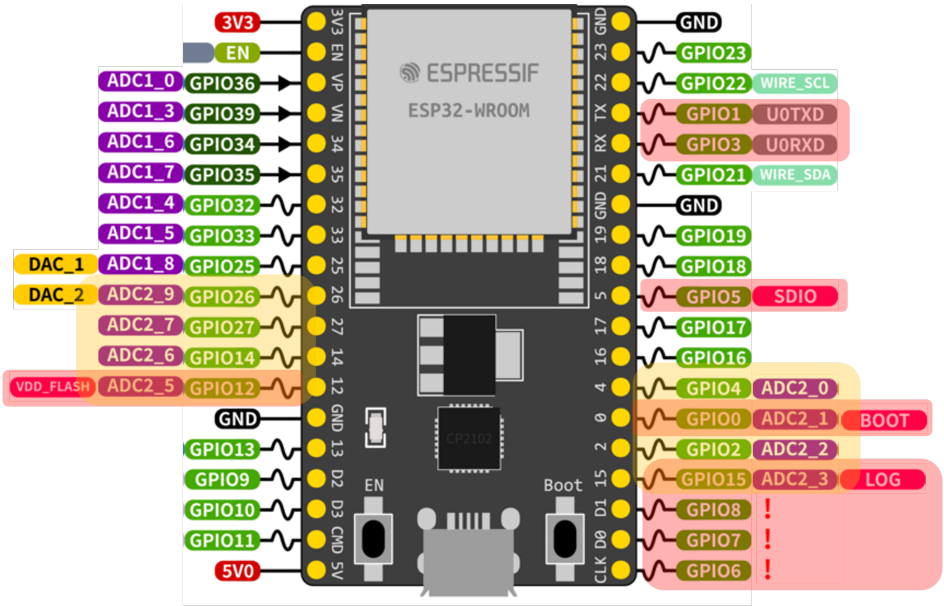


Figure: Carte ESP32 DevKit v1 (30 broches)

## Composants clés

- **ESP-WROOM-32** : Module principal
- **CP2102** : Convertisseur USB-UART
- **AMS1117** : Régulateur 3.3V
- **Boutons** :
  - EN (Reset)
  - BOOT (Flashage)
- **LEDs** :
  - Alimentation (rouge)
  - Utilisateur (bleue - GPIO2)

# Caractéristiques techniques

## Important

Variations entre modèles :

- Nombre de broches (30/36/...)
- Placement des composants
- Tension d'alimentation
- Fonctions des GPIO

**Toujours vérifier le datasheet!**

## Brochage type

| | |
|---|---|
| **3.3V** | Alimentation |
| **GND** | Masse |
| **GPIO0** | Boot mode |
| **GPIO2** | LED intégrée |
| **RX/TX** | UART |
| **SCL/SDA** | I2C |
| **MOSI/MISO** | SPI |
| **DAC** | Sortie analogique |

# ESP32 vs Arduino - Différences techniques

| **Caractéristique** | **ESP32** | **Arduino Uno** |
|---|---|---|
| Microcontrôleur | ESP32 (Xtensa Dual-Core) | ATmega328P (AVR 8-bit) |
| Fréquence | 240 MHz | 16 MHz |
| Mémoire Flash | 4MB (extensible) | 32KB |
| RAM | 520KB SRAM | 2KB SRAM |
| Connectivité intégrée | WiFi + Bluetooth | Aucune |
| GPIO | 34 broches | 14 broches |

## Avantages matériels ESP32

• 15x plus rapide • 250x plus de RAM • Connectivité native

# ESP32 vs Arduino - Aspects fonctionnels

| **Caractéristique** | **ESP32** | **Arduino** |
|---|---|---|
| Programmation | Arduino IDE • MicroPython • ESP-IDF | Arduino IDE (C/C++) |
| Écosystème | IoT • Réseau • Multitâche | Débutants • Prototypage rapide |
| Capteurs intégrés | Touch • Hall • Température | Aucun |
| Alimentation | 2.3-3.6V (Low Power Modes) | 5V (Linéaire) |
| Prix moyen | 5-10€ | 20-25€ |

## Complexité

- ESP32 : Gestion avancée (dual core, BLE stack)
- Arduino : Configuration minimaliste

# Quand choisir ?

## Préférer Arduino

- Projets débutants/simples
- Contraintes temps réel basiques
- Budget serré (clones low-cost)
- Éducation électronique de base

## Préférer ESP32

- Connexion Internet/Bluetooth
- Traitement multitâche
- Capteurs avancés
- Projets IoT professionnels

**Exemple Arduino** Capteur DHT22 + Écran LCD
**Exemple ESP32** Station météo connectée MQTT + App mobile

# Environnements de développement principaux

## Outils officiels

- Arduino IDE (avec core ESP32)
- ESP-IDF (Framework natif)
- Espressif IoT Development Framework
- Flash Download Tools

## Outils tiers

- PlatformIO (VSCode/Atom)
- MicroPython
- Simulateurs (Wokwi, QEMU)
- VS Code + Extensions

# Arduino IDE - Intégration classique

- Configuration simple via Boards Manager
- Syntaxe familière (setup/loop)
- Gestion des bibliothèques intégrée
- Débogage limité (Serial Monitor)

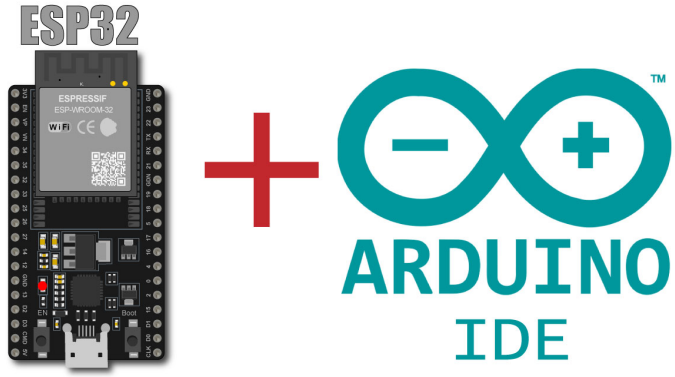


Figure: ESP32 dans Arduino IDE

# Arduino IDE - Intégration classique


Figure: Configuration ESP32 dans Arduino IDE

# PlatformIO - Environnement professionnel

Figure: Workflow PlatformIO dans VS Code

## Fonctionnalités clés

- Gestion avancée des dépendances
- Débogage avec JTAG/SWD
- Tests unitaires intégrés
- Multi-plateforme (Windows/Linux/macOS)
- Intégration CI/CD

## Avantages

- Meilleure gestion des gros projets
- Support multi-cartes

# ESP-IDF - Framework natif

## Composants principaux

- Outils de build basés sur CMake
- Pilotes matériels optimisés
- FreeRTOS intégré
- Outils de profiling mémoire
- Configuration menuconfig

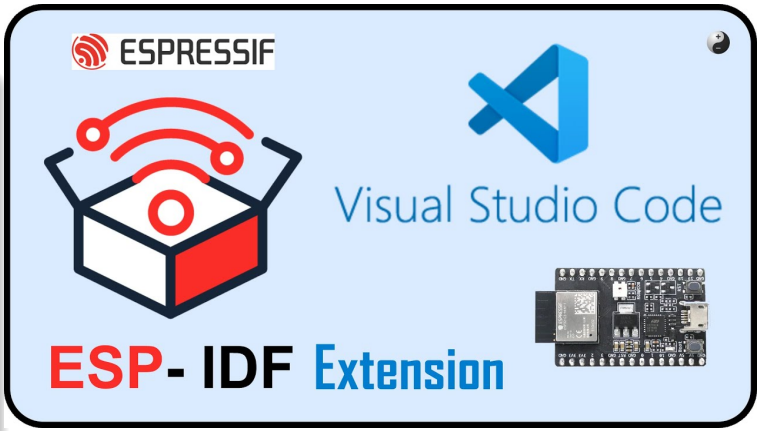


Figure: Output de compilation ESP-IDF

## Workflow typique

```
$ idf.py set-target esp32
$ idf.py menuconfig
$ idf.py build
$ idf.py -p PORT flash monitor
```

# MicroPython - Développement rapide

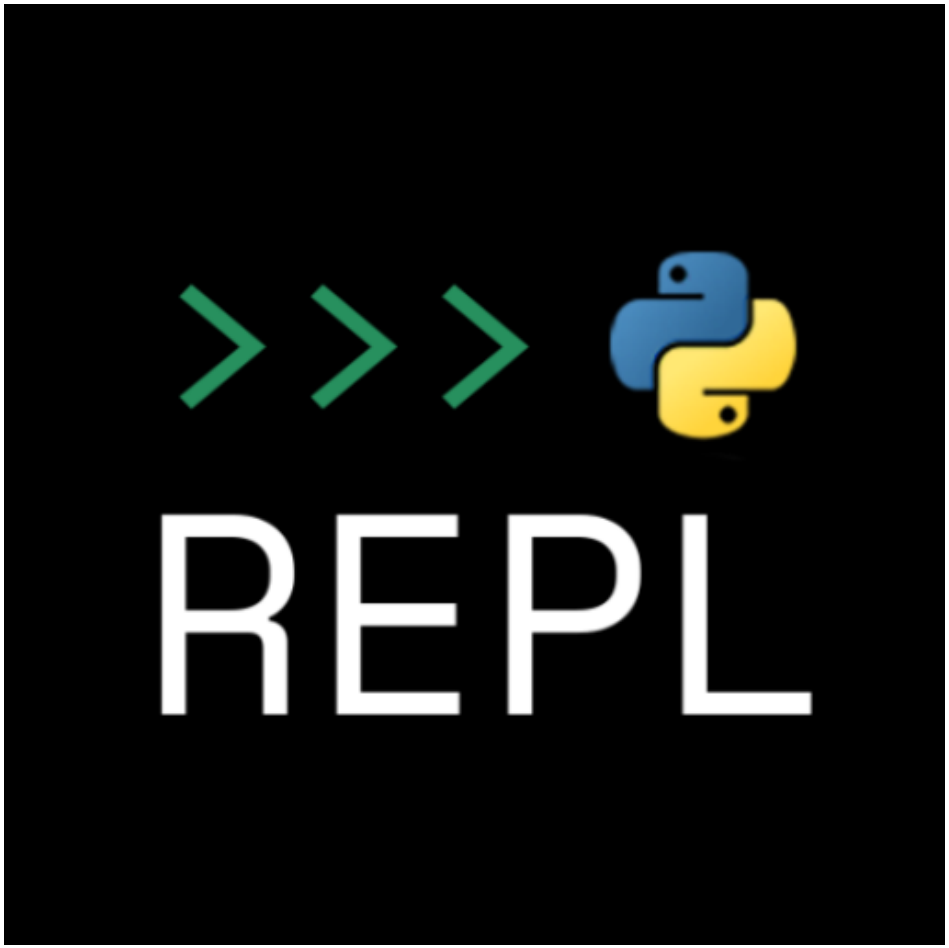


Figure: REPL MicroPython

- Interpréteur Python 3.4
- Accès direct au matériel
- Développement interactif

**Outils associés**

Thonny IDE • rshell • mpfshell

# Wokwi - Simulateur IoT en ligne

## Caractéristiques clés

- Simulation navigateur sans installation
- Support complet ESP32 (WiFi/BLE/MQTT)
- Composants virtuels :
  - Capteurs (DHT22, HC-SR04...)
  - Actionneurs (moteurs, relais)
  - Écrans (OLED, LCD TFT)
  - Protocoles (I2C, SPI, UART)
- Débogage temps réel avec Serial Monitor
- Collaboration en temps réel (partage de lien)

## Avantages

- Prototypage rapide
- Aucun matériel requis
- Intégration GitHub

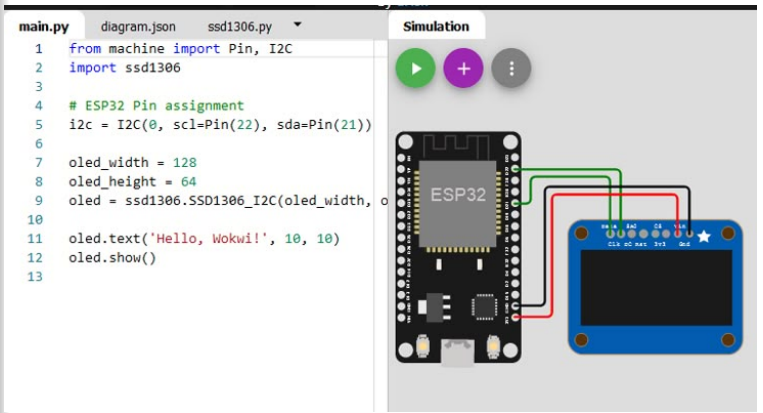


Figure: Interface Wokwi avec ESP32

# Introduction à Wokwi

- Plateforme en ligne de simulation pour IoT et systèmes embarqués
- Supporte ESP32, STM32, Arduino, Raspberry Pi Pico, etc.
- Simulateur de capteurs, moteurs, écrans et WiFi

# Étape 1 : Créer un compte Wokwi

- Accéder au site web de Wokwi
- Cliquer sur "Sign in" puis suivre le processus d'inscription


Figure: Page d'inscription Wokwi

# Étape 2 : Options d'inscription

- Choix de connexion via :
  - Compte Google
  - Compte GitHub
  - Email standard

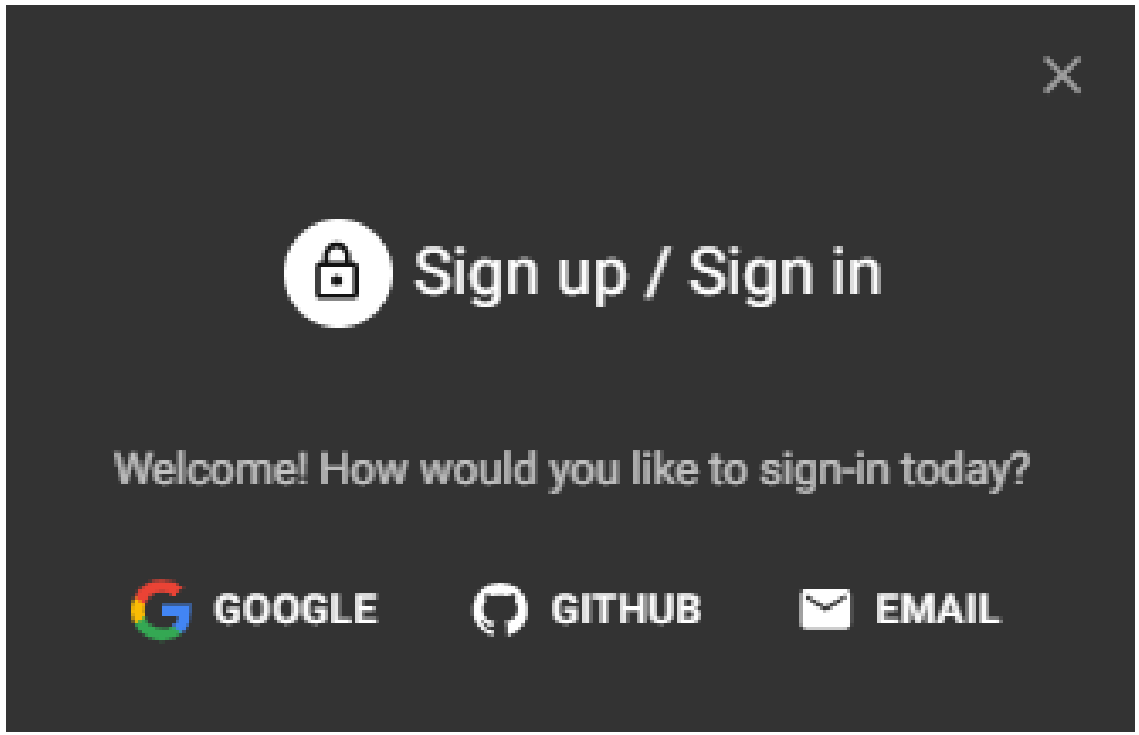


Figure: Processus d'inscription

# Étape 3 : Choix de la carte

- Sélectionner un contrôleur dans la liste
- Pour ceci : choisir ESP32


Figure: Menu de sélection des cartes

# Étape 4 : Modèle de base

- Naviguer dans les templates
- Sélectionner :
  - ESP32 → Starter templates → ESP32

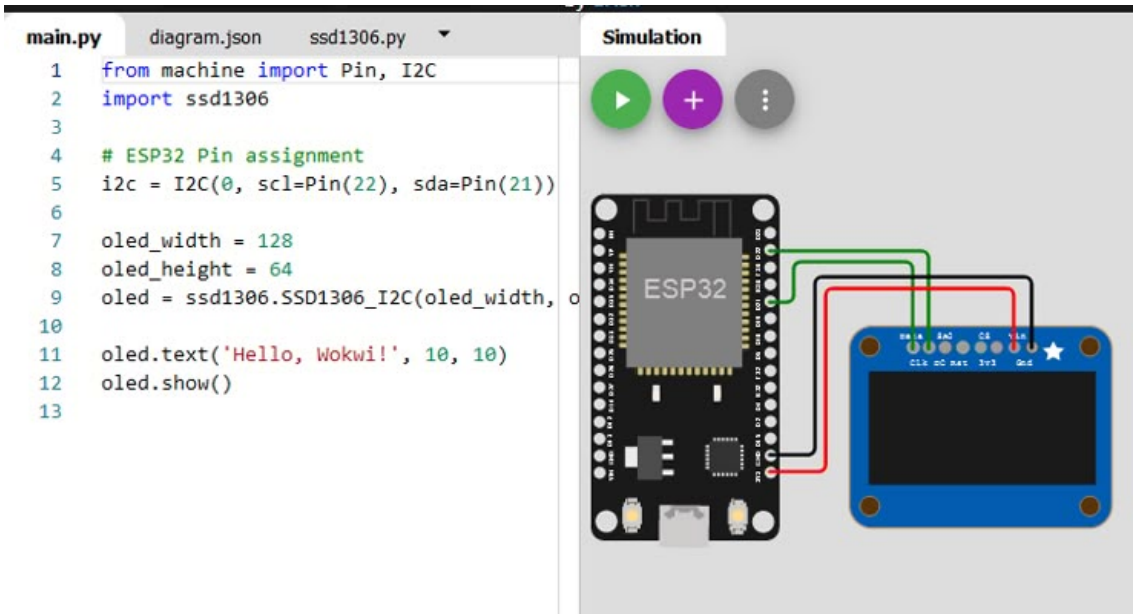


Figure: Template ESP32 de base

# Prochaines étapes

- Explorer l'interface de simulation
- Ajouter des composants électroniques
- Programmer et tester le système

**Prêt à simuler !**