🛠️ TP 1 — Module 3

Entrées / Sorties Numériques

PlatformIO · Arduino C++ · GPIO · LED · Bouton

📝 Changelog — V0.1.0

- Deck 3 : Prise en main PlatformIO, GPIO, boucle Arduino, exemple LED + bouton.

VS IDE Arduino vs PlatformIO

Critère	Arduino IDE	PlatformIO (VS Code)
Gestion des librairies	Manuel	`platformio.ini` automatique
Completion de code	Basique	IntelliSense complet
Débogage	Serial.print	Débogueur pas-à-pas
Gestion de projets	1 fichier	Projet structuré avec src/lib/test
Contrôle de version	Difficile	Git natif
Multi-cartes	1 à la fois	Plusieurs targets

> *PlatformIO = **l'IDE professionnel** pour l'embarqué.*

📁 Structure d'un projet PlatformIO

⚙️ Le fichier `platformio.ini`

```ini
[env:esp32dev]
platform  = espressif32
board     = esp32dev
framework = arduino

; Vitesse du moniteur série
monitor_speed = 115200

; Dépendances (bibliothèques)
lib_deps =
    bblanchon/ArduinoJson @ ^6.21.0
    knolleary/PubSubClient @ ^2.8

; Filtres de moniteur série (décodage des exceptions)
monitor_filters = esp32_exception_decoder
```

Un seul fichier remplace des dizaines de clics dans l'IDE. 🎯

🔄 La boucle Arduino : setup() + loop()

📌 Les fonctions GPIO essentielles

```
// Configurer une broche
pinMode(pin, OUTPUT);      // Sortie (LED, relais, moteur)
pinMode(pin, INPUT);       // Entrée (capteur, inter)
pinMode(pin, INPUT_PULLUP);// Entrée avec résistance tirage interne

// Écrire une valeur
digitalWrite(pin, HIGH);   // +3.3V sur la broche
digitalWrite(pin, LOW);    // 0V sur la broche

// Lire une valeur
int val = digitalRead(pin); // Retourne HIGH (1) ou LOW (0)

// PWM (simulation analogique)
analogWrite(pin, 128);      // 50% duty cycle (0-255)
```

🔌 Le Pull-up interne : INPUT_PULLUP

💡 Premier programme complet

```
#include <Arduino.h>

const int LED_PIN = 26;
const int BTN_PIN = 12;

void setup() {
    Serial.begin(115200);             // Démarrage moniteur série
    pinMode(LED_PIN, OUTPUT);         // LED en sortie
    pinMode(BTN_PIN, INPUT_PULLUP);   // Bouton avec pull-up interne
    Serial.println("Démarrage OK !");
}

void loop() {
    if (digitalRead(BTN_PIN) == LOW) { // LOW = bouton pressé (pull-up)
        digitalWrite(LED_PIN, HIGH);   // Allumer la LED
        Serial.println("Bouton pressé !");
    } else {
        digitalWrite(LED_PIN, LOW);    // Éteindre la LED
    }
}
```

🔍 Analysons le code

```
if (digitalRead(BTN_PIN) == LOW) {
```

Pourquoi LOW = pressé ?

Avec `INPUT_PULLUP`, la broche est maintenue à **3.3V (HIGH)** par la résistance interne.
Quand le bouton est pressé → court-circuit vers **GND (LOW)**.

> *Convention inversée : `LOW` signifie **actif**.*
> *C'est la norme industrielle pour les boutons.*

📟 Le Moniteur Série

Votre outil de débogage numéro 1 sur MCU :

```
Serial.begin(115200);          // Initialiser à 115200 bauds
Serial.println("Texte");       // Message + saut de ligne
Serial.print(variable);        // Afficher une variable
Serial.printf("T=%d°C\n", t); // Format printf
```

Dans PlatformIO :

- Raccourci : `Ctrl + Alt + S` pour ouvrir le moniteur
- Ou : icône 🔌 en bas de l'écran
- Assurez-vous que `monitor_speed = 115200` dans `platformio.ini`

⏱️ Timer non-bloquant

Ne jamais utiliser `delay()` dans un vrai projet — ça **bloque** toute la boucle !

```
// ❌ Mauvaise pratique : delay() bloque tout
digitalWrite(LED_PIN, HIGH);
delay(500);   // ← le programme est figé pendant 500ms
digitalWrite(LED_PIN, LOW);

// ✅ Bonne pratique : millis() non-bloquant
unsigned long previousMs = 0;
const long interval = 500;

void loop() {
    unsigned long currentMs = millis();
    if (currentMs - previousMs >= interval) {
        previousMs = currentMs;
        // Basculer la LED
        digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    }
    // Le reste du programme peut tourner librement ici !
}
```

🏋️ Exercice 1 — Clignotement non-bloquant

Objectif : Faire clignoter une LED à 2 Hz (500ms ON / 500ms OFF) **sans delay()**, pendant que le Serial affiche un compteur toutes les secondes.

Attendu :

1. LED clignote régulièrement
2. Moniteur série affiche : `Compteur : 1`, `Compteur : 2`...
3. Appuyer sur le bouton change la fréquence de clignotement (100ms ↔️ 500ms)

Aide : Utilisez **deux timers** basés sur `millis()`.

🏋️ Exercice 2 — Morse SOS

Objectif : Programmer la LED pour émettre **SOS en code Morse** en boucle.

```
S = ···   (3 points courts : 200ms ON / 200ms OFF)
O = ——— (3 traits longs  : 600ms ON / 200ms OFF)
S = ···   (3 points courts)
         [pause 1 seconde entre chaque SOS]
```

Contrainte : Zéro `delay()`. Utilisez une machine à états et `millis()`.

🎯 Synthèse du TP 1

Concept	Retenir
`setup()`	Initialisation unique au démarrage
`loop()`	Boucle infinie, jamais bloquante idéalement
`pinMode()`	Configurer direction de la broche
`INPUT_PULLUP`	LOW = actif (bouton pressé)
`millis()`	Timer non-bloquant (jamais `delay()`)
`Serial.println()`	Débogage via port série

Prochain TP → Programmation Orientée Objet pour structurer le code embarqué

👨‍💻 Questions ?

TP 1 — Entrées/Sorties Numériques

Réda BOUREBABA & Sébastien Antonico