📡 Module 6

Capteurs, Signaux et Réseau

Lecture analogique · Debouncing · Wi-Fi · MQTT

📝 Changelog — V0.1.0

- Deck 6 : Traitement du signal, anti-rebond, Wi-Fi ESP32, protocoles IoT, MQTT.

⚡ Le monde physique n'est pas binaire

En informatique classique : `0` ou `1`, propre et précis.
En électronique physique : les signaux **fluctuent**, **rebondissent**, **varient** en continu.

Trois défis du traitement du signal embarqué :

1. **Lecture analogique** — convertir une tension continue en valeur numérique
2. **Debouncing** — filtrer les rebonds mécaniques d'un bouton
3. **Filtrage** — lisser les valeurs bruyantes d'un capteur

📊 Lecture Analogique (ADC)

Sur l'ESP32, l'ADC (*Analog-to-Digital Converter*) convertit une tension en entier :

```
// Résolution 12 bits → 0 à 4095
int rawValue = analogRead(A0);  // Entrée 0-3.3V → 0-4095

// Conversion en tension
float voltage = rawValue * (3.3f / 4095.0f);

// Conversion en pourcentage (ex: potentiomètre)
float percent = (rawValue / 4095.0f) * 100.0f;

Serial.printf("Brut: %d | Tension: %.2fV | %%: %.1f%%\n",
              rawValue, voltage, percent);
```

Capteurs analogiques typiques : potentiomètre, LDR (lumière), NTC (température), microphone.

⚠️ Limites de l'ADC ESP32

```
// Attention : l'ADC ESP32 n'est PAS linéaire sur toute la plage !
// 0.0V → 0.1V : zone morte (lectures instables)
// 3.2V → 3.3V : saturation (lectures imprécises)

// Plage utile recommandée : 0.1V - 3.1V (valeurs 150-3800 environ)

// Astuce : appliquer un filtre passe-bas logiciel
float filtered = 0;
void loop() {
    int raw = analogRead(A0);
    // Filtre exponentiel (alpha = 0.1 = "lent", 0.9 = "rapide")
    filtered = 0.9f * filtered + 0.1f * raw;
}
```

> *L'ADC de l'ESP32 est notoirement non-linéaire. Pour une précision métrologique, utilisez un ADS1115 (ADC externe 16 bits via I2C).*

Le problème des rebonds (Bounce)

Quand on appuie sur un bouton mécanique, les contacts se **rebondissent** électriquement :

Debouncing logiciel

```
class Button {
  private:
    int  pin;
    bool lastStableState   = HIGH;
    bool lastRawState      = HIGH;
    unsigned long lastChange = 0;
    const unsigned long DEBOUNCE_MS = 50;

  public:
    Button(int p) : pin(p) {}
    void begin() { pinMode(pin, INPUT_PULLUP); }

    bool wasClicked() {
        bool current = digitalRead(pin);
        if (current != lastRawState) {      // Changement détecté
            lastChange   = millis();         // Mémoriser l'instant
            lastRawState = current;
        }
        if ((millis() - lastChange) > DEBOUNCE_MS) {  // Signal stable ?
            if (current != lastStableState && current == LOW) {
                lastStableState = current;
                return true;                // Clic validé !
            }
            lastStableState = current;
        }
        return false;
    }
};
```

📡 Modes Wi-Fi de l'ESP32

L'ESP32 supporte **3 modes Wi-Fi** :

Mode	Description	Usage
STA (Station)	Connexion à un routeur existant	Objet connecté au réseau local
AP (Access Point)	Crée son propre réseau Wi-Fi	Configuration, démo sans routeur
STA+AP	Les deux simultanément	Portail captif de configuration

```
#include <WiFi.h>

// Mode STA : connexion au réseau local
WiFi.mode(WIFI_STA);
WiFi.begin("NomWiFi", "MotDePasse");
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println(WiFi.localIP()); // Affiche l'IP obtenue
```

🔗 Connexion Wi-Fi robuste

```
void connectWiFi() {
    WiFi.begin(SSID, PASSWORD);
    int attempts = 0;
    while (WiFi.status() != WL_CONNECTED && attempts < 20) {
        delay(500);
        Serial.print(".");
        attempts++;
    }
    if (WiFi.status() == WL_CONNECTED) {
        Serial.printf("\n✅ Connecté ! IP : %s\n",
                      WiFi.localIP().toString().c_str());
    } else {
        Serial.println("\n❌ Échec de connexion Wi-Fi");
        ESP.restart(); // Redémarrer et réessayer
    }
}
```

📬 MQTT : le standard de l'IoT

HTTP : lourd, connexion à chaque requête, conçu pour le web.
MQTT : léger, **persistant**, conçu pour les objets contraints.

Modèle Publish / Subscribe :

- Les objets **publient** des messages sur des *topics*
- Les abonnés **reçoivent** automatiquement les messages
- Un **broker** central centralise les échanges

> *MQTT = "Le facteur de l'IoT"* 📬

🌐 Architecture MQTT Publish/Subscribe

📋 Topics MQTT : convention de nommage

```
# Structure hiérarchique avec /
maison/salon/temperature       → 22.5
maison/salon/lumiere/etat      → ON
capteurs/usine/machine1/vibration → 0.42

# Wildcards (côté abonné seulement)
maison/+/temperature    → + correspond à UN niveau
maison/#                → # correspond à TOUS les sous-niveaux
```

Bonnes pratiques :

- Lowercase, pas d'espaces, pas de caractères spéciaux
- Hiérarchie : `[lieu]/[pièce]/[device]/[mesure]`
- Préfixer par le projet : `monapp/...`

💻 MQTT avec PubSubClient

```
#include <WiFi.h>
#include <PubSubClient.h>

WiFiClient   wifi;
PubSubClient mqtt(wifi);

void callback(char* topic, byte* payload, unsigned int len) {
    // Appelé à chaque message reçu
    String msg = String((char*)payload).substring(0, len);
    Serial.printf("📨 [%s] : %s\n", topic, msg.c_str());
    if (String(topic) == "maison/lumiere") {
        if (msg == "ON")  led.on();
        if (msg == "OFF") led.off();
    }
}

void setup() {
    mqtt.setServer("192.168.1.100", 1883); // Adresse du broker
    mqtt.setCallback(callback);
}
```

💻 Publication et reconnexion MQTT

```
void reconnectMQTT() {
    while (!mqtt.connected()) {
        Serial.print("Connexion MQTT...");
        if (mqtt.connect("ESP32-Salon", "user", "pass")) {
            Serial.println(" OK !");
            mqtt.subscribe("maison/lumiere"); // S'abonner au topic
        } else {
            Serial.printf(" Erreur %d, nouvel essai dans 5s\n",
                          mqtt.state());
            delay(5000);
        }
    }
}

void loop() {
    if (!mqtt.connected()) reconnectMQTT();
    mqtt.loop(); // Traiter les messages entrants

    // Publier la température toutes les 30 secondes
    if (publishTimer.isExpired()) {
        float temp = readTemperature();
        char msg[20];
        snprintf(msg, sizeof(msg), "%.1f", temp);
        mqtt.publish("maison/salon/temperature", msg);
        publishTimer.start(30000);
    }
}
```

🏗️ Architecture IoT complète

🔒 Sécuriser MQTT

```
// MQTT sans TLS = données en clair sur le réseau !

// Utiliser MQTT over TLS (port 8883)
#include <WiFiClientSecure.h>
WiFiClientSecure wifiSecure;
PubSubClient    mqttSecure(wifiSecure);

// Charger le certificat CA du broker
wifiSecure.setCACert(root_ca_cert);

// Authentification par certificat client
wifiSecure.setCertificate(client_cert);
wifiSecure.setPrivateKey(client_key);

// Connexion sur le port TLS
mqttSecure.setServer("broker.example.com", 8883);
```

🏋️ TP Réseau : Capteur connecté

Objectif : publier la température d'un DHT22 sur MQTT toutes les 30 secondes.

Matériel : ESP32 + capteur DHT22 (ou simulation avec potentiomètre)

Tâches :

1. Configurer Wi-Fi en mode STA
2. Connecter au broker MQTT (ex: `test.mosquitto.org`)
3. Publier sur `iot-cours/[votre_prenom]/temperature`
4. S'abonner à `iot-cours/[votre_prenom]/led` pour piloter une LED à distance
5. **(Bonus)** Afficher l'état sur un écran OLED (I2C)

🎯 Synthèse du Module 6

Concept	Retenir
`analogRead()`	Convertit 0-3.3V en 0-4095 (12 bits)
Debouncing	Ignorer les changements < 50ms
Wi-Fi STA	Connexion au réseau local
MQTT topic	Hierarchie `lieu/pièce/device/mesure`
Publish	Envoyer une valeur sur un topic
Subscribe	Recevoir les messages d'un topic

Prochain module → Projet Final — La Porte de Garage

🧑‍💻 Questions ?

Module 6 — Capteurs, Signaux et Réseau
Réda BOUREBABA & Sébastien Antonico