Laboratoires IoT avec PlatformIO

SmartComputerLab

Contenu

0. **Introduction** **2**
0.1 ESP32 Soc – une unité avancée pour les architectures IoT 3
0.2 Carte LOLIN D32 3
0.3 IoT DevKit une plate-forme de développement IoT 4
0.4 L'installation de PlatfomIO 5
0.4.1 Installation de VSCode 5
0.4.2 Installer le package PlatformIO IDE pour VSCode 5
0.4.3 Premier démarrage de PlatformIO sur VSCode 6
0.4.4 Le menu PIO 7
0.4.5 Créer un nouveau projet (ESP32, ESP8266, …) 8
0.4.6 Décryptage du fichier platformio.ini 10
0.4.7 Edition du code 11
0.7.8 Premier exemple 12
Laboratoire 1 ..... **15**
1.1 Premier exemple – l'affichage des données 15
A faire: 16
1.2 Deuxième exemple – capture et affichage des valeurs 17
1.2.1 Capture de la température/humidité par SHT21 17
A faire: 18
1.2.2 Capture de la luminosité par BH1750 19
1.2.3 Capture de la luminosité par MAX44009 20
1.2.5 Capture de la pression/température avec capteur BMP180 21
1.2.6 Capture de présence avec un capteur PIR SR602 22
Laboratoire 2 ..... **23**
Communication en WiFi et broker MQTT 23
2.1 Client MQTT – envoi et réception des messages 23
A faire : 25
2.2 Simple broker MQTT avec ESP8266 26
A faire : 28
Laboratoire 3 – WiFi avec WiFiManager et serveur ThingSpeak.com (.fr) **29**
3.1 Introduction 29
3.1.1 Un programme de test – scrutation du réseau WiFi 29
3.2 Mode WiFi – STA, client WEB et serveur ThingSpeak 31
A faire 32
Laboratoire 4 – communication longue distance avec LoRa (*Long Range*) **39**
4.1 Introduction 39
4.1.1 Modulation LoRa 39
4.1.2 Paquets LoRa 40
4.2 Premier exemple – émetteur et récepteur des paquets LoRa 41
A faire : 43
4.3 onReceive() – récepteur des paquets LoRa avec une interruption 44
A faire : 45
Laboratoire 5 - Développement de simples passerelles IoT **46**
5.1 Passerelle LoRa-ThingSpeak 46
5.1.1 Le principe de fonctionnement 46
5.1.2 Les éléments du code 46
5.1.3 Code complet pour une passerelle des paquets en format de base 49
A faire : 50
5.2 Passerelle LoRa – WiFi – broker MQTT 51
5.2.1 L'émetteur des messages MQTT sur LoRa 51
5.2.2 La passerelle des messages MQTT sur LoRa vers WiFi et broker MQTT 52
A faire : 53

Laboratoire 2

Communication en WiFi et broker `MQTT`

MQTT est l'un des protocoles IoT les plus importants, largement utilisé dans les projets IoT pour connecter des cartes telles que ESP8266, ESP32 au courtier (**broker**) cloud MQTT.
Cette introduction suppose que vous connaissez MQTT et que vous connaissez les aspects de base tels que le courtier MQTT, les rubriques (**topics**) MQTT et l'architecture de publication (**publish**) et d'abonnement (**subscribe**).

Si vous débutez dans ce protocole IoT, il est utile d'en savoir plus sur **MQTT** en lisant une description technique.
Ce laboratoire explique comment développer un client ESP32 MQTT pour publier des messages MQTT et s'abonner aux rubriques MQTT. Nous allons utiliser la bibliothèque **`MQTT.h`** pour connecter l'ESP32 au courtier (**broker**) **MQTT**. Pour mieux comprendre comment utiliser un client ESP32 MQTT, nous utiliserons un exemple de publication ESP32 MQTT se connectant au courtier **`broker.emqx.io`** .

Le client enverra les données dans les rubriques (topics) MQTT. De plus, nous allons découvrir comment s'abonner aux rubriques MQTT pour que l'ESP32 puisse recevoir des données.

2.1 Client MQTT – envoi et réception des messages

Voici un exemple complet du code qui permet de se connecter au broker, d'envoyer – publier et recevoir - s'abonner à un **topic** donnée. Le code fonctionne sur la carte ESP32.

```
#include <Arduino.h>

#include <WiFi.h>
#include <MQTT.h>
#include <Wire.h>
#include "SSD1306Wire.h"
#include "SSD1306.h"
SSD1306Wire display(0x3c, 12, 14); // SDA, SCL

 const char* ssid = "PhoneAP";
 const char* pass = "smartcomputerlab";

// const char* ssid = "MyMQTT-5";
// const char* pass = "smartcomputerlab";

const char* mqttServer = "broker.emqx.io";
//const char* mqttServer = "192.168.5.1";

WiFiClient net;
MQTTClient client;

typedef struct
{
char topic[32];
char mess[32];
} MQTTpack; // MQTT packet

void display_MQTT(char *top, char *mes)
{
char buff[32];
display.init();
display.flipScreenVertically();
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.setFont(ArialMT_Plain_10);
display.drawString(8, 0, "SmartComputerLab");
display.drawString(0, 14, "Topic");
display.drawString(8, 26, top);
display.drawString(0, 40, "Message");
display.drawString(8, 52, mes);
display.display();
}
```

```
void connect() {
char cbuff[128];
Serial.print("checking wifi...");
while (WiFi.status() != WL_CONNECTED) {
Serial.print(".");
delay(1000);
}
Serial.print("\nconnecting...");
while (!client.connect("IoT.GW5")) {
Serial.print("."); delay(1000);
}
Serial.println("\nIoT.GW1 - connected!");
client.subscribe("/esp32/test");
}

void messageReceived(String &topic, String &payload)
{
MQTTpack rp; // received packet
Serial.println("incoming: " + topic + " - " + payload);
topic.toCharArray(rp.topic,32);payload.toCharArray(rp.mess,32);
display_MQTT(rp.topic,rp.mess);
}

void setup() {
Serial.begin(9600);
Wire.begin(12,14); // SCL,SDA
Serial.println();
display.init();
display.flipScreenVertically();
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.setFont(ArialMT_Plain_16);
display.drawString(0, 0, "Start MQTT");
display.display();
delay(2000);
WiFi.begin(ssid, pass);
client.begin(mqttServer, net);
client.onMessage(messageReceived);
connect();
}

int val=0;

void loop() {
char vbuff[15];
client.loop();
delay(10); // <- fixes some issues with WiFi stability
if (!client.connected()) { connect(); }
sprintf(vbuff,"%d",val);
client.publish("/esp32/test", vbuff);
Serial.printf("published to /esp32/test:%s\n",vbuff);
delay(6000);val++;
}
```

Il commence par l'inclusion des bibliothèques **`WiFi.h`** et **`MQTT.h`** et par les les déclarations des paramètres d'accès au **WiFi** et au broker **MQTT**.
Après la connexion de notre client MQTT – **`IoT.GW1`** au broker **MQTT** nous allons nous abonner (**subscribe**) aux plusieurs rubriques (**topics**).

Le **topic** de base est: **`/esp32/test`**

Dans la fonction d'initialisation **`setup()`** :

```
WiFi.begin(ssid, pass);
client.begin(mqttServer, net);
client.onMessage(messageReceived);
connect();
```

on initialise les paramètres (**credentials**) de WiFi et du client MQTT avec la fonction de **callback**-**`messageReceived()`** , puis on appelle la fonction connect() pour activer la connexion WiFi et s'abonner (**subscribe**) aux plusieurs **topic** et **sous-topic**.

Dans la boucle (tâche de fond) principale **`loop()`**, on vérifie la connexion **WiFi** et broker **MQTT**, puis on on publie la valeur du compteur (val) sur le **topic** - **`/esp32/test`**

A faire :

1. Charger l'application Android **`MyMQTT`** et la tester avec notre exemple.
2. Ajouter un capteur et envoyer les valeurs captées comme message MQTT
3. Séparer les fonctionnalites **publish** et **subscribe** sur **2 DevKits** : un DevKit envoie les message (publish) et l'autre les reçoit (subscribe) puis affiche sur son écran OLED.

2.2 Simple broker MQTT avec ESP8266

Le SoC **ESP8266** est le prédécesseur du SoC **ESP32**. Il contient un simple processeur Tensilica Xtensa LX106.
Un exemple de caractéristiques de ce SoC est indiqué ci-dessous.

- 32-bit RISC CPU: Tensilica Xtensa LX106, 80 MHz;
- 64 Kio de RAM instruction, 96 Kio de RAM data ;
- QSPI flash externe - 512 Kio à 4 Mio (supporte jusqu'à 16 Mio) ;
- IEEE 802.11 b/g/n Wi-Fi ;
    - TR switch intégré, balun, LNA, amplificateur de puissance et matching network ;
    - Authentification par WEP ou WPA/WPA2 ou bien réseau ouvert

- 16 broches GPIO
- Interfaces SPI, I2C;
- Interface I^2S avec DMA (partageant les broches avec les GPIO) ;
- UART sur des broches dédiées, plus un UART dédié aux transmission pouvant être géré par GPIO2;
- 1 10-bit ADC

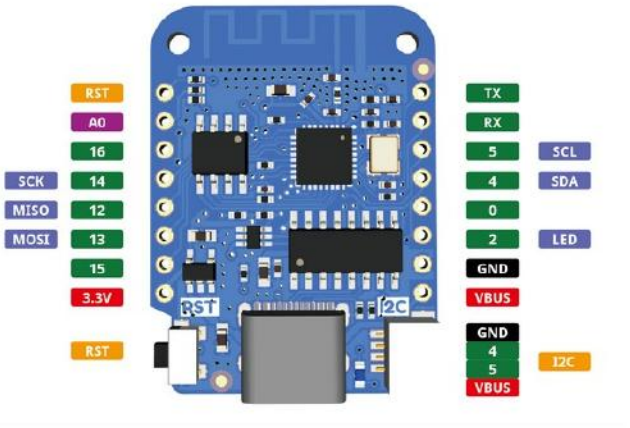

Fig 2.1 Les broches d'un ESP8266-Wemos D1 mini

Voici le fichier **`platformio.ini`** pour initialiser l'utilisation d'une carte **Wemos d1_mini_lite** et importer la bibliothèque **`uMQTTBroker.h`** .

```
[env:d1_mini_lite]
platform = espressif8266
board = d1_mini_lite
framework = arduino
lib_deps = https://github.com/martin-ger/uMQTTBroker.git
```

Le code complet du mini broker inclut les bibliothèques **`ESP8266WiFi.h`** et **`uMQTTBroker.h`** .
Ensuite nous déclarons les credentials , le nom du point d'accès et le mot de passe.
Le broker peut fonctionner sur le réseau local associé à un point d'accès donné (**`ssid,pass`**) , ou peut lui même devenir le point d'accès (**`ssidAP,pasAP`**).

La classe **`myMQTTBroker`** importée de la bibliothèque **`uMQTTBroker.h`** est **surchargée** par les méthodes de l'utilisateur, vous pouvez les modifier/compléter selon le besoin:

- **`onConnect`** – affichage de l'adresse IP
- **`onDisconnect`**
- **`onAuth`** – affichage du nom utilisateur et son mot de passe (optionnel)
- **`onData`** - affichage du **topic** et message associé

Les fonctions **`startWiFiClient()`** et **`startWiFiAP()`** permettent d'établir une connexion WiFi avec un point d'accès externe ou d'activer un propre point d'accès.

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
```

```
#include "uMQTTBroker.h"
#include <Wire.h> // Only needed for Arduino 1.6.5 and earlier
#include "SSD1306Wire.h" // legacy: #include "SSD1306.h"
SSD1306Wire display(0x3c, D2, D1); // SDA, SCL

float stab[2];
char ssid[] = "PhoneAP"; // your network SSID (name)
char pass[] = "smartcomputerlab"; // your network password
char ssidAP[] = "MyMQTT-6"; // softAP SSID (name)
char passAP[] = "smartcomputerlab"; // softAP network password
bool WiFiAP = true; // Do yo want the ESP as AP?

void display_SSD1306()
{
char buff[32];
display.init();
display.flipScreenVertically();
display.setTextAlignment(TEXT_ALIGN_LEFT);
display.setFont(ArialMT_Plain_16);
display.drawString(0, 0, " MyMQTT-6 ");
display.drawString(0, 24, " IP:192.168.5.1");
display.setFont(ArialMT_Plain_10);
display.drawString(8, 48, "SmartComputerLab");
display.display();
}

class myMQTTBroker: public uMQTTBroker
{
  public:
  virtual bool onConnect(IPAddress addr, uint16_t client_count)
  {
  Serial.println(addr.toString()+" connected");
  return true;
  }
virtual void onDisconnect(IPAddress addr, String client_id)
{
  Serial.println(addr.toString()+" ("+client_id+") disconnected");
}
virtual bool onAuth(String username, String password, String client_id)
{
  Serial.println("Username/Password/ClientId: "+username+"/"+password+"/"+client_id);
  return true;
}
virtual void onData(String topic, const char *data, uint32_t length)
  {
  char buf[24];
  char data_str[length+1];
  os_memcpy(data_str, data, length);
  data_str[length] = '\0';
  Serial.println("received topic '"+topic+"' with data '"+(String)data_str+"'");
  //printClients();
  }

// Sample for the usage of the client info methods
virtual void printClients() {
for (int i = 0; i < getClientCount(); i++)
        {
        IPAddress addr;
        String client_id;
        getClientAddr(i, addr);
        getClientId(i, client_id);
        Serial.println("Client "+client_id+" on addr: "+addr.toString());
        }
  }
};

myMQTTBroker myBroker;

void startWiFiClient()
{
Serial.println("Connecting to "+(String)ssid);
WiFi.mode(WIFI_STA);
```

```
WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    }
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: " + WiFi.localIP().toString());
}

void startWiFiAP()
{
WiFi.mode(WIFI_AP);
// softAP specific configuration
if(!WiFi.softAPConfig(IPAddress(192,168,5,1),IPAddress(192,168,5,1),IPAddress(255,255,255,0)))
  {
  Serial.println("AP Config Failed");
  }
WiFi.softAP(ssidAP, passAP);
Serial.println("AP started");
Serial.println("IP address: " + WiFi.softAPIP().toString());
}

void setup()
{
Serial.begin(9600);
Serial.println();
Wire.begin(D1,D2); // SCL,SDA
Serial.println();
display_SSD1306();
Serial.println();
Serial.println("Initialized");
// Start WiFi
if (WiFiAP)
       startWiFiAP();
else
       startWiFiClient();
// Start the broker
Serial.println("Starting MQTT broker");
myBroker.init();
// Subscribe to anything to monitor the received messages
myBroker.subscribe("#");
}

int counter = 0;

void loop()
{
// Publish the counter value as String
//myBroker.publish("broker/counter", (String)counter++);
// wait 10 seconds
delay(10000);
}
```

Ci-dessous l'exemple d'exécution do broker en mode **`softAP`** avec l'adresse IP par défaut **`192.168.4.1`**

```
IP address: 192.168.4.1
Starting MQTT broker
received topic 'broker/counter' with data '0'
received topic 'broker/counter' with data '1'
received topic 'broker/counter' with data '2'
received topic 'broker/counter' with data '3'
received topic 'broker/counter' with data '4'
received topic 'broker/counter' with data '5'
```

A faire :

1. Tester l'exemple du broker ci-dessus
2. Utiliser **2 DevKits pour tester notre broker**: un **DevKit** envoie les message (publish) et l'autre les reçoit (subscribe) puis affiche sur son écran OLED.