Laboratoires IoT avec PlatformIO

SmartComputerLab

Contenu

Laboratoire 3 – WiFi avec WiFiManager et serveur `ThingSpeak.com (.fr)`

3.1 Introduction

Dans ce laboratoire nous allons nous intéresser aux moyens de la **communication** et de la **présentation** (**stockage**) des résultats. Etant donné que le SoC ESP32 intègre les modems de WiFi et de Bluetooth nous allons étudier la communication par **WiFi**.

Principalement nous avons deux modes de fonctionnement **WiFi** – mode station **`STA`**, et mode point d'accès **`softAP`**. Dans le mode **`STA`** nous pouvons connecter notre carte à un point d'accès WiFi (qui peut être notre smartphone). Dans le mode **`AP`** nous créons un point d'accès sur la carte ESP32. Cet point d'accès peut être utilisé pour effectuer une configuration de la carte, par exemple en lui fournissant le nom du point d'accès (**`ssid`**) et le mot de passe pour ce point d'accès.
Un troisième mode appelé **`ESP-NOW`** permet de communiquer entre les cartes directement (sans un Point d'Accès), donc plus rapidement et à plus grande distance, par le biais des trames physiques **MAC**.
Une fois la communication WiFi est opérationnelle nous pouvons choisir un des multiples protocoles pour transmettre nos données : **`UDP`**, **`TCP`**, **`HTTP/TCP`**, .
Par exemple la carte peut fonctionner comme client ou serveur WEB.

3.1.1 Un programme de test – scrutation du réseau WiFi

Pour commencer notre étude de la communication WiFi essayons un exemple permettant de scruter notre environnement dans la recherche de point d'accès visibles par notre modem.

```
#include "WiFi.h"
#include <Wire.h>
#include "SSD1306Wire.h"
SSD1306Wire display(0x3c, 12, 14);

void display_SSD1306(char *text)
{
  char buff[256];
  strcpy(buff,text);
  display.init();
  display.flipScreenVertically();
  display.setTextAlignment(TEXT_ALIGN_LEFT);
  display.setFont(ArialMT_Plain_10);
  display.drawString(0, 0, "ETN - WiFi SCAN");
  display.drawString(0, 14, buff);
  display.display();
}

void setup()
{
  char buf[16];
  Serial.begin(9600);
  Wire.begin(12,14);
  Serial.println();
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(2000);
  Serial.println("Setup done");
}

void loop()
{
  int apmax=8, num;
  char buf[16];
  char tab[256];

  Serial.println("scan start");
  Serial.println("WiFi scan");
  // WiFi.scanNetworks will return the number of networks found
  int n = WiFi.scanNetworks();
  Serial.println("WiFi scan");
    if (n == 0) {
        Serial.println("no networks found");
```

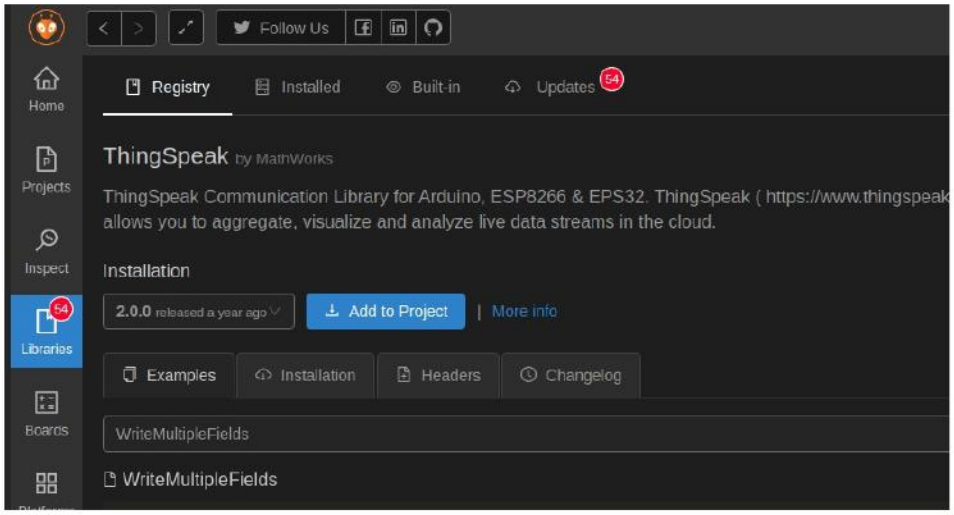
```
    } else {
        //Serial.print(n);
        sprintf(buf,"WiFi AP#%d\n",n);
        Serial.println(buf);
        delay(2000);
        if(n<apmax) num=n; else num=apmax;
        //Serial.println(" networks found");
        for (int i = 0; i < num; ++i) {
            // Print SSID and RSSI for each network found
            Serial.print(i + 1);
            Serial.print(": ");
            WiFi.SSID(i).toCharArray(buf,16);
            Serial.print(buf);
            if(!i) strcpy(tab,buf);else strcat(tab,buf);strcat(tab,"\n");
            display_SSD1306(tab);
            //Serial.println(WiFi.SSID(i));
            Serial.print(" (");
            Serial.print(WiFi.RSSI(i));
            Serial.print(")");
            Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
            delay(10);
        }
    }
    Serial.println("");
    delay(5000);
}
```

A faire

Tester et analyser le programme ci-dessus.

3.2 Mode WiFi – **STA**, client WEB et serveur **ThingSpeak**

Une connexion WiFi permet de créer une application avec un client WEB. Ce client WEB peut envoyer les requêtes HTTP vers un serveur WEB.
Dans nos laboratoires nous allons utiliser un serveur IoT externe **ThingSpeak.com**.
ThingSpeak est un serveur «*open source*» qui peut être installé sur un PC ou sur une carte SBC.
Nous pouvons donc envoyer les requêtes HTTP (par exemple en format **GET** et les données attachées à **URL**) sur le serveur **ThingSpeak.com**.
La préparation de ces requêtes peut être laborieuse, heureusement il existe une bibliothèque **ThingSpeak.h** qui permet de simplifier cette tâche. Il faut donc installer la bibliothèque **ThingSpeak.h**.



Après l'inscription sur le serveur **ThingSpeak.com** vous créez un ***channel*** composé de max 8 ***fields***. Ensuite vous pouvez envoyer et lire vos données dans ce ***fields*** à condition de fournir la clé d'écriture associée au ***channel***.
Une écriture/envoi de plusieurs valeurs en virgule flottante est effectuée comme suit :

```
ThingSpeak.setField(1, sensor[0]);   // préparation du field1
ThingSpeak.setField(2, sensor[1]);   // préparation du field2
```

puis

```
ThingSpeak.writeFields(myChannelNumber[1], myWriteAPIKey[1]);   // envoi
```

Voici le début d'un tel programme :

```
#include <WiFi.h>
#include "ThingSpeak.h"
char ssid[] = "PhoneAP";                 //  your network SSID (name)
char pass[] = "smartcomputerlab";        // your network passw
unsigned long myChannelNumber = 1;
const char * myWriteAPIKey="MEH7A0FHAMNWJE8P" ;
WiFiClient  client;
void setup()
{
Serial.begin(9600);
WiFi.disconnect(true);  // effacer de l'EEPROM WiFi credentials
delay(1000);
WiFi.begin(ssid, pass);
delay(1000);
    while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
    }
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip); Serial.println("WiFi setup ok");
  delay(1000);
  ThingSpeak.begin(client); // connexion (TCP) du client au serveur
  delay(1000);
  Serial.println("ThingSpeak begin");
}
```

Pour simplifier l'exemple dans la fonction de la boucle principale **`loop()`** nous allons envoyer les données d'un compteur.

```
int tout=10000;  // en millisecondes
float luminosity=100.0, temperature=10.0;

void loop()
{
ThingSpeak.setField(1, luminosity);  // préparation du field1
ThingSpeak.setField(2, temperature);  // préparation du field1
    while (WiFi.status() != WL_CONNECTED) {
    delay(500);Serial.print(".");
    }
ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
luminosity++;temperature++;
delay(tout);
}
```

Attention : pour le serveur **`ThingSpeak.com`** utilisé gratuitement (max 1 an) la **valeur minimale** de **`tout`** est 20 secondes.

Figure 2.1. **ThingSpeak** : l'affichage des données envoyées sur le serveur type **ThingSpeak**

A faire

1. Créer un canal **`ThingSpeak`** puis récupérer les paramètres du canal : *number* et *write key*.
2. Intégrer l'utilisation des capteurs de Température/Humidité et de la Luminosité.
3. Envoyer ces données cycliquement (par exemple toutes les 30 seconde) vers le serveur **`ThingSpeak`**.

3.3 Mode WiFi – `STA` et `WiFiManager`

La carte ESP32 permet de fonctionner en mode Point d'Accès qui permet de déployer un simple serveur WEB avec une page de configuration des paramètres sur la carte ESP32.
Parmi ces paramètres il y a la possibilité de proposer (et enregistrer) le nom d'un point d'accès et son mot de passe.

`WiFiManager` est une fonction qui réalise ce type d'opérations.

Voici le contenu de **`platformio.ini`** avec la bibliothèque **`WiFiManager`**

```
[env:lolin_d32]
platform = espressif32
board = lolin_d32
framework = arduino
lib_deps =
https://github.com/tzapu/WiFiManager.git
```

et le programme de test pour l'utilisation du **`WiFiManager`**.

```
#include <WiFiManager.h> // https://github.com/tzapu/WiFiManager

void setup() {
    WiFi.mode(WIFI_STA); // set mode to STA+AP
    Serial.begin(9600);
    WiFiManager wm;
    //reset settings - wipe credentials for testing
    wm.resetSettings();  // à tester , puis à commenter
    // Automatically connect using saved credentials,
    bool res;
    res = wm.autoConnect("ESP32AP",NULL); // no password
    if(!res) {
        Serial.println("Failed to connect"); // ESP.restart();
    }
    else {
        //if you get here you have connected to the WiFi
        Serial.println("connected...yeey :)");
    }
}

void loop() { }
```

Sur votre terminal IDE **`WiFiManager`** affiche ses paramètres et l'état de fonctionnement.

Vous devrez vous connecter avec votre smartphone sur son pont d'accès (ici) ESP32AP et aller avec votre navigateur sur **`192.168.4.1`** pour accéder à la page d'ac**cue**il , puis choisir votre point d'accès local et fournir le mot de passe.
Le point d'accès et le mot de passe seront enregistrés sur votre carte.
N'oubliez pas der **commenter** la ligne :

```
wm.resetSettings();
```

pour pouvoir garder ces valeurs en mémoire **EEPROM** du ESP32.

Attention:

Sur certaines cartes ESP32 il faut effacer intégralement la mémoire flash avant le chargement du code avec l'utilitaire **`esptool.py`**:

```
~/.platformio/packages/framework-espidf/components/esptool_py/esptool/esptool.py

~/arduino-1.8.1$ ./hardware/espressif/esp32/tools/esptool.py  -p /dev/ttyUSB0 -b 460800  erase_flash
```

Voici le déroulement d'affichage de **`WiFiManager`** dans le cas de connexion sur un nouveau point d'accès.

```
*WM: [3] WIFI station disconnect
*WM: [3] WiFi station enable
*WM: [2] Disabling STA
```

```
*WM: [2] Enabling AP
*WM: [1] StartAP with SSID:  ESP32AP
*WM: [2] AP has anonymous access!
*WM: [1] SoftAP Configuration
*WM: [1] --------------------
*WM: [1] ssid:             ESP32AP
*WM: [1] password:
*WM: [1] ssid_len:         7
*WM: [1] channel:          1
*WM: [1] authmode:
*WM: [1] ssid_hidden:
*WM: [1] max_connection:   4
*WM: [1] country:          CN
*WM: [1] beacon_interval:  100(ms)
*WM: [1] --------------------
*WM: [1] AP IP address: 192.168.4.1
*WM: [3] setupConfigPortal
*WM: [1] Starting Web Portal
*WM: [3] dns server started with ip:  192.168.4.1
*WM: [2] HTTP server started
*WM: [2] WiFi Scan completed in 5509 ms
*WM: [2] Config Portal Running, blocking, waiting for clients...

*WM: [3] WIFI station disconnect
*WM: [3] WiFi station enable
*WM: [2] Disabling STA
*WM: [2] Enabling AP
*WM: [1] StartAP with SSID:  ESP32AP
*WM: [2] AP has anonymous access!
*WM: [1] SoftAP Configuration
*WM: [1] --------------------
*WM: [1] ssid:             ESP32AP
*WM: [1] password:
*WM: [1] ssid_len:         7
*WM: [1] channel:          1
*WM: [1] authmode:
*WM: [1] ssid_hidden:
*WM: [1] max_connection:   4
*WM: [1] country:          CN
*WM: [1] beacon_interval:  100(ms)
*WM: [1] --------------------
*WM: [1] AP IP address: 192.168.4.1
*WM: [3] setupConfigPortal
*WM: [1] Starting Web Portal
*WM: [3] dns server started with ip:  192.168.4.1
*WM: [2] HTTP server started
*WM: [2] WiFi Scan completed in 5509 ms
*WM: [2] Config Portal Running, blocking, waiting for clients...

WM: [2] NUM CLIENTS: 0
*WM: [3] -> connectivitycheck.platform.hicloud.com
*WM: [2] <- Request redirected to captive portal
*WM: [2] NUM CLIENTS: 1
*WM: [2] <- HTTP Root
*WM: [3] -> 192.168.4.1
*WM: [3] lastconxresulttmp: WL_IDLE_STATUS
*WM: [3] lastconxresult: WL_DISCONNECTED
*WM: [2] WiFi Scan completed in 5106 ms
*WM: [2] <- HTTP Wifi
*WM: [2] Scan is cached 7 ms ago
*WM: [1] 18 networks found
*WM: [2] DUP AP: FreeWifi_secure
*WM: [2] DUP AP: SFR WiFi Mobile
*WM: [2] DUP AP: Livebox-C82A
*WM: [2] AP:  -52 DIRECT-G8M2070 Series
*WM: [2] AP:  -60 orange
*WM: [2] AP:  -62 Livebox-08B0
*WM: [2] AP:  -63 VAIO-MQ35AL
*WM: [2] AP:  -75 Livebox-08B0_Ext
*WM: [2] AP:  -84 FreeWifi_secure
*WM: [2] AP:  -85 Livebox-3D36
*WM: [2] AP:  -88 SFR WiFi Mobile
*WM: [2] AP:  -90 SFR WiFi FON
```

```
*WM: [2] AP:  -93 Livebox-F936
*WM: [2] AP:  -93 Livebox-C82A
*WM: [2] AP:  -94 SFR_E0B0
*WM: [2] AP:  -95 SFR_A0A0
*WM: [2] AP:  -95 freebox_ODTMTH
*WM: [2] AP:  -96 Bbox-EA1EB632
*WM: [3] lastconxresulttmp: WL_IDLE_STATUS
*WM: [3] lastconxresult: WL_DISCONNECTED
*WM: [3] Sent config page
*WM: [3] -> connectivitycheck.platform.hicloud.com
*WM: [2] <- Request redirected to captive portal
*WM: [2] <- HTTP WiFi save
*WM: [3] Method: POST
*WM: [3] Sent wifi save page
*WM: [2] processing save
*WM: [2] Connecting as wifi client...
*WM: [3] STA static IP:
*WM: [2] setSTAConfig static ip not set, skipping
*WM: [1] CONNECTED:
*WM: [1] Connecting to NEW AP: Livebox-08B0
*WM: [3] Using Password: G79ji6dtEptVTPWmZP
*WM: [3] WiFi station enable
*WM: [1] connectTimeout not set, ESP waitForConnectResult...
*WM: [2] Connection result: WL_CONNECTED
*WM: [3] lastconxresult: WL_CONNECTED
*WM: [1] Connect to new AP [SUCCESS]
*WM: [1] Got IP Address:
*WM: [1] 192.168.1.20
*WM: [2] disconnect configportal
dhcps: send_nak>>udp_sendto result 0
*WM: [2] restoring usermode STA
*WM: [2] wifi status: WL_CONNECTED
*WM: [2] wifi mode: STA
*WM: [1] config portal exiting
connected...yeey :)
*WM: [3] unloading
```

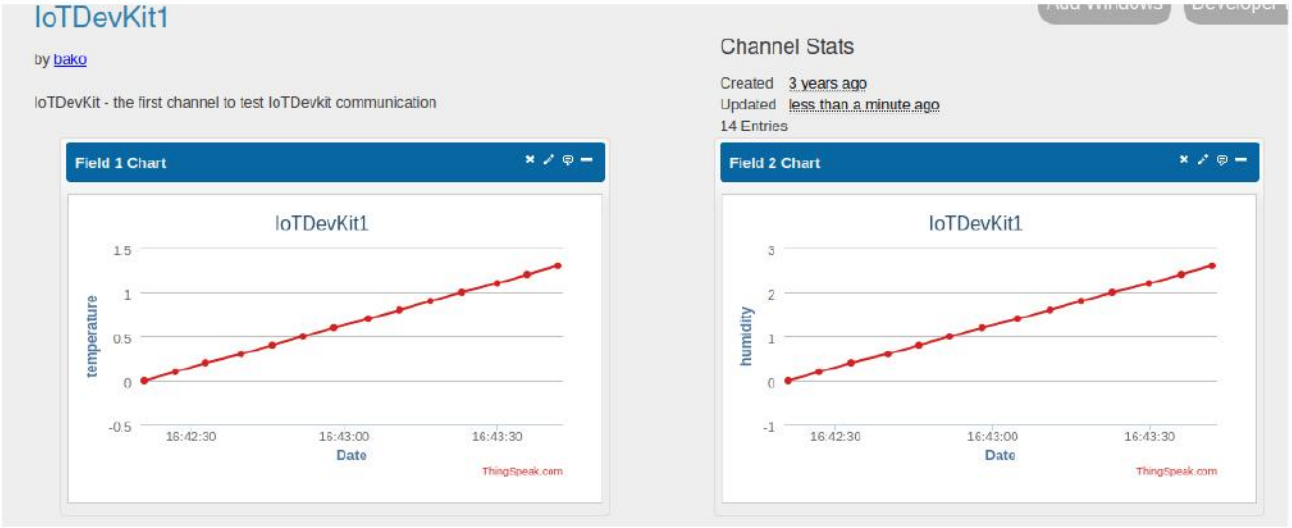
3.4 Envoi des données sur ThingSpeak avec WiFiManager

```
#include <Arduino.h>
#include <WebServer.h>
#include <WiFiManager.h>
#include "ThingSpeak.h"
char ssid[] = "Livebox-08B0"; // your network SSID (name)
char pass[] = "G79ji6dtEptVTPWmZP"; // your network passw
unsigned long myChannelNumber = 1538804;
const char * myWriteAPIKey="YOX31M0EDKO0JATK" ;
int dout=15;
WiFiClient client;

void setup() {
WiFi.mode(WIFI_STA);
Serial.begin(9600);
WiFiManager wm;
//wm.resetSettings();
bool res;
res = wm.autoConnect("ESP32AP",NULL); // password protected ap
if(!res) {
Serial.println("Failed to connect");  // ESP.restart();
}
else {
//if you get here you have connected to the WiFi
Serial.println("connected...yeey :)");
}
ThingSpeak.begin(client); // connexion (TCP) du client au serveur
delay(1000);
Serial.println("ThingSpeak begin");
}

float temperature=0.0,humidity=0.0;

void loop() {
Serial.println("Fields update");
ThingSpeak.setField(1, temperature);
ThingSpeak.setField(2, humidity);
ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
delay(dout*1000);
temperature+=0.1;
humidity+=0.2;
}
```



3.5 Réception des données de ThingSpeak

```
#include <Arduino.h>
#include <WebServer.h>
#include <WiFiManager.h>
#include "ThingSpeak.h"
char ssid[] = "Livebox-08B0"; // your network SSID (name)
char pass[] = "G79ji6dtEptVTPWmZP"; // your network passw
unsigned long myChannelNumber = 1538804;
const char * myWriteAPIKey="YOX31M0EDKO0JATK" ;

WiFiClient client;

void setup() {
WiFi.mode(WIFI_STA);
Serial.begin(9600);
WiFiManager wm;
//reset settings - wipe credentials for testing
// wm.resetSettings();
bool res;
res = wm.autoConnect("ESP32AP",NULL); // password protected ap
if(!res) {
Serial.println("Failed to connect");
// ESP.restart();
}
else {
//if you get here you have connected to the WiFi
Serial.println("connected...yeey :)");
}
ThingSpeak.begin(client); // connexion (TCP) du client au serveur
delay(1000);
Serial.println("ThingSpeak begin");
}

float temperature=0.0,humidity=0.0;
float tem,hum;

void loop() {
Serial.println("Fields update");
ThingSpeak.setField(1, temperature);
ThingSpeak.setField(2, humidity);
ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);
delay(6000);
temperature+=0.1;
humidity+=0.2;
tem =ThingSpeak.readFloatField(myChannelNumber,1);
// if channel is private you need to add the reda key: AVG5MID7I5SPHIK8
tem =ThingSpeak.readFloatField(myChannelNumber,1,"20E9AQVFW7Z6XXOM");
Serial.print("Last temperature:");
Serial.println(tem);
delay(6000);
hum =ThingSpeak.readFloatField(myChannelNumber,2,"20E9AQVFW7Z6XXOM");
Serial.print("Last humidity:");
Serial.println(hum);
delay(15000);
}
```

Voici l'affiche correspondant à l'exécution de ces programme :

```
*WM: [3] STA static IP:
*WM: [2] setSTAConfig static ip not set, skipping
*WM: [1] Connecting to SAVED AP: Livebox-08B0
*WM: [3] Using Password: G79ji6dtEptVTPWmZP
*WM: [3] WiFi station enable
*WM: [1] connectTimeout not set, ESP waitForConnectResult...
*WM: [2] Connection result: WL_CONNECTED
*WM: [3] lastconxresult: WL_CONNECTED
*WM: [1] AutoConnect: SUCCESS
*WM: [1] STA IP Address: 192.168.1.20
connected...yeey :)
```

```
ThingSpeak begin
*WM: [3] unloading
Fields update
Last temperature:0.00
Last humidity:0.00
Fields update
Last temperature:0.10
Last humidity:0.20
Fields update
Last temperature:0.20
Last humidity:0.40
Fields update
Last temperature:0.30
Last humidity:0.60
Fields update
Last temperature:0.40
```

A faire (comme dans l'exemple précédent) :

1. Créer un canal **ThingSpeak** puis récupérer les paramètres du canal : *number* , *read key* et *write key*.
2. Intégrer l'utilisation des capteurs de Température/Humidité et de la Luminosité.

Envoyer ces données cycliquement (par exemple toutes les 30 seconde) vers le serveur **ThingSpeak**.