UE INF-IoT
« Internet of Things »

Sujet de TP : Reconfiguration d'une platine à micro-contrôleur via Wi-Fi

L'objectif de cette séance de travaux pratiques est d'étudier comment une platine portant un micro-contrôleur ESP32 (ou équivalent) peut être configurée depuis un équipement mobile (par exemple un smartphone), via une liaison Wi-Fi.

1 Motivation

Dans le TP précédent vous avez développé et déployé sur ESP32 un programme permettant de faire de la collecte périodique de données à partir de capteurs, et la transmission de ces données vers un serveur distant, via une liaison Wi-Fi. Ce programme gérait la découverte des points d'accès Wi-Fi environnants, la connexion à l'un de ces points d'accès, et la transmission de données proprement dite. Le SSID et le mot de passe du réseau Wi-Fi choisi, de même que l'adresse et le n° de port du serveur visé, étaient fixés « en dur » dans le code source du programme. Cette façon de faire peut convenir pour un simple test, mais elle ne peut être généralisée : il faudrait pouvoir modifier les paramètres de fonctionnement du système (pour changer aisément de réseau Wi-Fi ou de serveur) sans avoir à recompiler et recharger le code.

Pour ce faire, l'une des option possibles, que nous allons explorer dans cette séance de TP, consiste à faire temporairement de notre platine ESP32 un point d'accès Wi-Fi hébergeant un mini-serveur Web. En se connectant à ce point d'accès Wi-Fi avec un smartphone, par exemple, un utilisateur peut accéder au serveur Web, et remplir un formulaire dont les champs correspondent aux paramètres que l'on souhaite modifier. Les nouveaux paramètres fournis par l'utilisateur sont alors enregistrés en mémoire Flash.

2 Montage

Aucun montage n'est requis pour cette séance de TP : vous vous contenterez d'utiliser une platine supportant un ESP32.

3 Développement d'un programme de test

3.1 Objectif

En utilisant l'éditeur VIsual Studio Code avec l'extension PlatformIO, vous allez développer un petit programme permettant :

1. de configurer l'ESP32 en tant que point d'accès logiciel *(Soft AP)* ;
2. d'héberger sur l'ESP32 un mini-serveur Web offrant sous forme de page Web un formulaire à renseigner pour modifier certains paramètres ;
3. de lire et écrire ces paramètres dans la mémoire Flash de l'ESP32.

Ce programme sera développé en utilisant le Framework Arduino, et notamment la librairie *arduino-esp32* [1]. Cette librairie contient des classes permettant d'activer un point d'accès logiciel, et d'accéder à la mémoire Flash (vue comme de la mémoire EEPROM [2]). Vous aurez également besoin d'installer la librairie *ESP32WebServer* développée par Ivan Grokhotkov.

1. `https://github.com/espressif/arduino-esp32`
2. La mémoire Flash est une variété particulière de mémoire EEPROM. On y accède par blocs (ou secteurs) plutôt que par mot-mémoire, contrairement aux autres types d'EEPROM.

3.2 Démarche

Pour vous faire gagner du temps, le squelette du programme que vous devez réaliser est mis à votre disposition. Dans ce squelette les fonctions `setup()` et `loop()` sont déjà définies, ainsi que quelques fonctions annexes dont vous aurez besoin.

Les différents paramètres pouvant être modifiés et enregistrés en mémoire Flash sont définis dans la structure `Settings`. Il s'agit de paramètres qui pourraient typiquement permettre à un programme de gérer la connection à un point d'accès Wi-Fi, puis l'accès à un serveur distant.

Le code gérant l'accès en lecture et écriture à la mémoire Flash (vue comme de l'EEPROM) est ici déjà écrit. Il vous reste donc à écrire de quoi gérer le mode Soft AP, et de quoi définir le comportement du serveur Web. Par défaut celui-ci retourne au client un formulaire Web (voir la fonction `handleRoot()` qui est déjà presque entièrement écrite), et enregistre les nouveaux paramètres saisis dans ce formulaire via la fonction `handleSet()`.

Lorsque votre programme sera au point, vous pourrez essayer de le combiner avec le programme de collecte de données développé dans le TP précédent. L'objectif est alors d'avoir un programme final dont le comportement normal est de faire de la collecte et de la transmssion de données, mais qui sur un événement précis (par exemple l'appui sur un bouton) peut basculer en mode « reconfiguration » : il bascule alors temporairement du mode « client Wi-Fi » au mode « point d'accès Wi-Fi », ce qui permet de le reconfiguer depuis un smartphone. Une fois cette reconfiguration effectuée, il rebascule en mode « client Wi-Fi », mais cette fois avec les nouveaux paramètres tels qu'ils ont été définis pendant la reconfiguration.

Squelette du programme

```
#include <Arduino.h>
#include <WiFi.h>
#include <EEPROM.h>
#include <ESP32WebServer.h>

// =============================================================================
// SETTINGS MANAGEMENT
// =============================================================================
#define EEPROM_ADDRESS        0
#define EEPROM_PARTITION_SIZE 128

#define COOKIE  3.14

struct Settings {
  float cookie;
  char ssid[32];
  char passwd[32];
  int channel;
  byte addr[4];
  int port;
};

Settings settings;

// =============================================================================

#define WEB_SERVER_PORT       80

ESP32WebServer web_server(WEB_SERVER_PORT);
WiFiClient client;
String mac;          // MAC address of the local device

// =============================================================================
// SETTINGS MANAGEMENT
// =============================================================================
void print_settings() {

  Serial.print("Cookie  : "); Serial.println(settings.cookie);
  Serial.print("SSID    : "); Serial.println(settings.ssid);
  Serial.print("Password: "); Serial.println(settings.passwd);
  Serial.print("Channel : "); Serial.println(settings.channel);
  Serial.print("Server  : ");
  Serial.print(settings.addr[0]); Serial.print(".");
  Serial.print(settings.addr[1]); Serial.print(".");
  Serial.print(settings.addr[2]); Serial.print(".");
  Serial.print(settings.addr[3]); Serial.print(":");
  Serial.println(settings.port);
}

// =============================================================================
void save_settings(int addr) {

  Serial.println("Storing settings in EEPROM");
  EEPROM.put(addr, settings);
  if (! EEPROM.commit())
    Serial.println("ERROR: failed to write to EEPROM");
  // Serial.println("Done.");
}

// =============================================================================
```

```
61  void read_settings(int addr) {
62
63    Serial.print("Reading settings from EEPROM: ");
64    EEPROM.get(addr, settings);
65
66    if (settings.cookie == (float)COOKIE)
67      Serial.println("OK");
68    else {
69      Serial.println("Failed (invalid cookie). Will use default values");
70      settings.cookie = (float)COOKIE;
71      strcpy(settings.ssid, "CHANGE_THIS");
72      strcpy(settings.passwd, "********");
73      settings.channel = 6;
74      settings.addr[0] = 10;
75      settings.addr[1] = 0;
76      settings.addr[2] = 0;
77      settings.addr[3] = 1;
78      settings.port    = 80;
79      save_settings(addr);
80    }
81
82    print_settings();
83  }
84
85  // ============================================================================
86  // Enable the EEPROM to access a small partition, and read settings from
87  // this partition
88  void setup_EEPROM() {
89
90    if (! EEPROM.begin(EEPROM_PARTITION_SIZE)) {
91      Serial.println("ERROR: failed to initialize EEPROM");
92      while(true);
93    }
94
95    read_settings(EEPROM_ADDRESS);
96  }
97
98  // ============================================================================
99  // WEB SERVER
100 // ============================================================================
101
102 void handleRoot() {
103
104   String msg = "<!DOCTYPE HTML>\n";
105   msg += "<html lang=\"en-US\">\n";
106   msg += "<body>\n";
107   msg += "<h1>ESP32 board</h1>";
108   msg += "<h2>MAC=" + mac + "</h2>\n";
109   msg += "<h2> <form action=\"/set\" method=\"get\">\n";
110   msg += "<p>SSID:<br>\n";
111   msg += "<input type=\"text\" name=\"ssid\" value=\"" + String(settings.ssid) + "\">\n";
112   msg += "<p>Password:<br>\n";
113   msg += "<input type=\"password\" name=\"passwd\" value=\"" + String(settings.passwd) + "\">
114   msg += "<p>Channel:<br>\n";
115   msg += "<input type=\"number\" name=\"channel\" style=\"width: 3em\" min=\"0\" max=\"13\" v
116   msg += "<p>Server (IP address):<br>\n";
117   msg += "<input type=\"number\" name=\"ad1\" style=\"width: 3em\" min=\"0\" max=\"255\" valu
118   msg += "<input type=\"number\" name=\"ad2\" style=\"width: 3em\" min=\"0\" max=\"255\" valu
119   msg += "<input type=\"number\" name=\"ad3\" style=\"width: 3em\" min=\"0\" max=\"255\" valu
120   msg += "<input type=\"number\" name=\"ad4\" style=\"width: 3em\" min=\"0\" max=\"255\" valu
121   msg += "<p>Port:<br>\n";
122   msg += "<input type=\"number\" name=\"port\" style=\"width: 5em\" min=\"0\" max=\"65535\" v
123   msg += "<p>\n";
```

```
124     msg += "<button type=\"submit\">Submit</button>\n";
125     msg += "</form></h2>\n";
126     msg += "</body>\n</html>\n";
127 
128     Serial.println("TO BE COMPLETED");
129 }
130 
131 // ================================================================================
132 void handleSet() {
133 
134     Serial.println("TO BE COMPLETED");
135 }
136 
137 // ================================================================================
138 void handleNotFound() {
139 
140     Serial.println("TO BE COMPLETED");
141 }
142 
143 // ================================================================================
144 // Initialize and start the Web server
145 void setup_Web_server() {
146 
147     Serial.println("TO BE COMPLETED");
148 }
149 
150 // ================================================================================
151 // Wzit for a Web client, process this client's request, then return
152 void handle_Web_client() {
153 
154     Serial.println("TO BE COMPLETED");
155 }
156 
157 
158 // ================================================================================
159 // ACCESS POINT
160 // ================================================================================
161 
162 // ================================================================================
163 // Enable the software access point mode on this device
164 //
165 // This AP will use address 10.0.0.1, and deliver addresses in range
166 // 10.0.0.0/24 to host stations.
167 //
168 // The SSID broadcast by the AP will be of the form 'ESP32_xxx', where 'xxx'
169 // is the MAC address of the device.
170 void setup_AP() {
171 
172     Serial.println("TO BE COMPLETED");
173 }
174 
175 // ================================================================================
176 void setup() {
177 
178     // Initialization of the serial console
179     Serial.begin(115200);
180 
181     setup_EEPROM();
182 
183     setup_AP();
184 
185     setup_Web_server();
186 
```

```
187     Serial.println(); Serial.println();
188   }
189
190   // ================================================================================
191   void loop() {
192
193     handle_Web_client();
194   }
```