Institut Galilée

24/04/2018

# IoT-Labs-2018


## Groupe :

**Romain CARREY:** carrey.romain@gmail.com

**Magilan ELILTHAMILVALAVAN:** magilan_95@hotmail.fr

**Leila ABDELLI :** infotelecom2@gmail.com

## Professeurs encadrants :

**Hamidi MASSINISSA :** massinissa.hamidi@lipn.univ-paris13.fr

**Aomar OSMANI :** aomar.osmani@lipn.univ-paris13.fr


Master 1 Informatique 2017-2018

# Table des matières

# Introduction

## Contexte :

Lors de ce projet il nous a été demandé de réaliser le code qui permettra de récupérer les données (température pression et humidité) d'un capteur (BME280) dans un microcontrôleur à puce (ESP32) puis d'envoyer ces mêmes données par wifi dans un serveur qui sera dans le même réseau local, cette communication sera basée sur la pile LwIP (Light-weight Internet Protocol).

## Objectifs :

Notre projet consistera à remplir les différents fichiers qui composent le code pour réaliser ce projet.

- Comprendre le code et son architecture des fichiers donnés.
- Remplir les 'TO DO' et les 'ENOSYS' (les fonctions non implémenté).
- Faire les extensions demandées (après avoir terminé ce qui était initialement demandé).

# I. Démarrage

## Prérequis :

Pour développer des applications pour ESP32, vous devez avoir :

- PC chargé avec Windows, Linux ou Mac
- Toolchain permettant de créer l'application pour l'ESP32 (ici Xtensa)
- ESP-IDF (framework) qui contient essentiellement l'API pour ESP32 et les scripts pour utiliser la Toolchain
- Un éditeur de texte pour écrire des programmes (Projets) en C, par ex. Éclipse
- La carte ESP32 elle-même et un câble USB pour la connecter au PC

De plus, dans notre cas il nous faut un capteur afin récupérer les données (BME280) pour les envoyés sur l'ESP32.

## Installation complète :

1) Prérequis

   Pour compiler avec ESP-IDF nous devons obtenir les paquets suivants :

   ```
   sudo apt-get install gcc git wget make libncurses-dev flex bison gperf python python-serial
   ```

2) Configuration de la Toolchain

   Nous avons téléchargé la Toolchain Xtensa selon la configuration de notre ordinateur (64-bits) sur lien suivant :

   http://www.lipn.univ-paris13.fr/~hamidi/misc/intro-iot/

(Nous avons téléchargé ‘xtensa-esp32-elf-linux64-1.22.0-61-gab8375a-5.2.0.tar.gz’)

Ensuite nous l’avons extrait donc le dossier ‘opt’ :

```
sudo tar −xvf
xtensa−esp32−elf−linux64−1.22.0−75−gbaf03c2−5.2.0.tar.gz
\ −C /opt
```

Enfin dans le fichier ‘.bashrc’ nous avons modifié le PATH afin de permettre à ‘xtensa-esp32-elf’ dans n’importe quel terminal :

```
export PATH="$PATH:$HOME/esp/xtensa-esp32-elf/bin"
```

3) Configuration du Framework ESP-IDF

Nous l’avons donc téléchargé avec la commande :

```
git clone --recursive
https://github.com/HamidiMassinissa/esp−idf
```

Puis comme pour la Toolchain nous avons modifié le PATH afin de permettre à ESP-IDF d’être accessible dans n’importe quel terminal :

```
export IDF_PATH=/path/to/the/location/of/your/repository
```

4) Autorisations pour communiquer avec l’ESP32

Etant donné qu’il est nécessaire d’avoir le droit de lecture et d’écriture sur l’ESP32 nous avons donc entré la commande :

```
sudo usermod −a −G dialout $USER
```

5) Python Package

Il nous manque que l’outil permettant de télécharger les exécutables sur la mémoire flash de notre carte de développement :

```
pip install esptool
```

6) Copie du code source

Il nous suffit enfin de copier le code de monsieur Massinissa présent sur son GitHub :

```
git clone https://github.com/HamidiMassinissa/iot-lab-2018.git
```

Et de télécharger les dernières mises à jour :

```
git pull origin master
```

7) Mise en place du serveur TCP

Pour faire des test le serveur TCP sera très utile. Pour le lancer il suffit d'écrire la commande :

```
node index.js
```

8) Lancement du serveur Json
   a. Prérequis
      Pour l'installer il suffit d'écrire la commande :
      ```
      npm install -g json-server
      ```
   b. Lancement
      Pour le lancer (ce qui nous permettra de recevoir des données sous format Json) il faut écrire :
      ```
      json-server --watch db.json
      ```

## Branchement

Pour savoir comment doivent se faire les branchements il suffit regarder les pins définis dans le fichier ‘sensor_reading.c’

```
32    #define PIN_NUM_MISO 19
33    #define PIN_NUM_MOSI 3
34    #define PIN_NUM_CLK  18
35    #define PIN_NUM_CS   5
```

Ainsi, nous devons relier le pin 19 de l’ESP32 sur le SDO du BME280, le Pin 3 sur le SDA, le pin 18 sur SCL, le pin 5 sur CSB, mais aussi nous devons brancher la pin VIN sur VCC et le pin GND sur GND.

# II. Partie technique

## Fonctionnement

Analysons techniquement les quatre fonctions appelées dans le main.

- « transmission_init() »

Elle permet d'initialiser la transmission (la queue) entre l'esp32 et le bme280 (afin d'échanger les données).

- « setup_connectivity() »

Cette fonction permettra à l'esp32 d'envoyer, de transmettre les données du bme280 vers le json-server.

Nous devons configurer 4 choses :

1) Phy_layer : Donc initialiser la partie physique. Ici la WIFI sera initialisée afin de se connecter au serveur.
   Il était possible d'utiliser le bluetooth par exemple mais nous n'avons pas eu le temps de le faire.

2) Net_layer : Nous ne devions rien toucher dans ce fichier. Il permettait de démarrer le protocol réseau utilisé (dans notre cas IPV4).

3) Trans_layer : Les fonctions dans ce fichier permettent d'établir la connexion vers le serveur. Nous pouvions choisir (switch) avec quel Protocol (TCP dans notre cas), en précisant le hostname et le port qu'écoute le serveur.
   C'est aussi ici qu'est envoyé la requête http vers le serveur.

4) App_layer : C'est ici qu'est créé la requête http. Requête qui contient les données que nous avions récupérées et préalablement formattées en JSON (nous verrons ça plus tard).
La requête est créée grâce à « snprintf ». Cette fonction créée l'entête de la requête http et y ajoute à la fin le reading déjà formaté.

- « setup_sensor() »

Cette partie va permettre de récupérer les données du bme280 et de les mettre dans la queue de transmission.
Dans cette fonction est initialisé le 'sensor' donc le bme280. Mais la fonction 'perform_sensor_reading()' est aussi appelée. Et c'est elle qui, grâce à trois autres fonctions, va permettre de récupérer les données du bme280, en créer un reading, et va le mettre dans la queue de transmission.

- « perform_transmission() »

Cette fonction va, elle, permettre de formatter en JSON.

Tout d'abord, elle va récupérer le reading qui est dans la queue de transmission. Et ensuite elle va le formater. Comment ?

Une première fonction va créer le formatted_reading, donc le mettre au format JSON (grâce à la library cJSON).
Et une deuxième va mettre ce 'formatted_reading' dans un tableau (representation) qui, lui, sera envoyé dans la requête http.

## Problèmes rencontrés

Il a été difficile de complètement se lancer dans le projet. Les monde des IOT nous était totalement inconnu, et nous ne maitrisions pas assez le langage C pour tout comprendre instantanément.

Nous avons donc eu plusieurs petits problèmes, notamment de compréhensions, durant ce projet.
Mais nous avons eu 3 problèmes majeurs :

1. Le premier gros problème est arrivé lorsque nous nous occupions de « trans_layer.c ». Majoritairement un problème de compréhension.

Nous devions établir la connexion sur un serveur TCP (de test dirons-nous) en utilisant la library Lwip. Comprendre ce qu'était le hostname, comment fonctionnait la connexion, et autre, était difficiles.

Et après avoir compris le principe, le souci a été de bien utiliser les fonctions et structures Lwip (notamment 'netconn_gethostByName' et la structure 'u_addr').

2. Le second problème concernait le formatage en JSON. Nous avons passé énormément de temps à trouver la solution au problème. Le problème concernait les pointeurs dans la fonction « reading_formatting() ».
En effet, au démarrage, nous ne comprenions pas pourquoi la requête http ne voulais pas s'envoyer. Et nous avions vite compris que le souci venait du fait que le tout était mal formaté. Mais, nous avions un printf qui affichait le bon formattage dans la fonction json_of_reading(). C'est la que nous avions compris que le problème venait des pointeurs (enfin surement) dans la fonction reading_formatting().

Et il a fallut du temps avant de comprendre qu'utiliser memcpy() était plus simple. Malgré tout, un « syntax error » s'affichait sur le terminal du serveur. Nous n'avons donc pas trouvé d'autre solution que de changer le type de la fonction « json_of_reading() » ( int8_t -> cJSON* ).

3. Le dernier problème était en réalité de comprendre le principe du snprintf() et de bien créer l'entête et de l'associer au formatted_reading. La solution était simple. Elle venait du fait qu'il fallait utiliser strlen() pour le 'content_length'. Mais il nous a fallut du temps avant d'y penser.

Avant ça, nous avions eu plusieurs légers soucis. Mais ils venaient surtout du fait que nous ne comprenions pas le projet.

# Conclusion

Avant de débuter ce projet, nous n'avions pas fait de C depuis longtemps. De plus, le monde de l'internet des objets nous était totalement inconnus ; et nous ne maitrisions pas vraiment les connexions client/serveurs en C.

Cependant, ce projet nous aura permis de beaucoup chercher par nous-même. Nous avons dû faire un gros travail de compréhension, tout en améliorant nos capacités de développeurs.

Ce travail nous sera donc bien évidemment bénéfique. Pour notre culture informatique dans un premier temps mais aussi, et surtout, pour notre avenir professionnel.