Marc NG KON TIA
Kajanan KANDAVEL

Internet Of Things

Projet

Introduction :

Ce module Internet Of Things a pour but de nous faire découvrir l'univers des objets connectés, qui nous est complètement étranger à la base. Nous avons suivi des cours magistraux, puis effectué des TP afin d'enrichir notre culture de l'IoT.

Ce rapport a pour but de détailler le contenu des TP réalisés au cours de ce module, et le projet final qui en découle. Ce rapport se décompose en 3 parties, suivant la progression des séances de TP : lab1, lab2 puis le projet.

Le lab1 concerne l'installation des outils de développement permettant de manipuler ensuite notre ESP32.

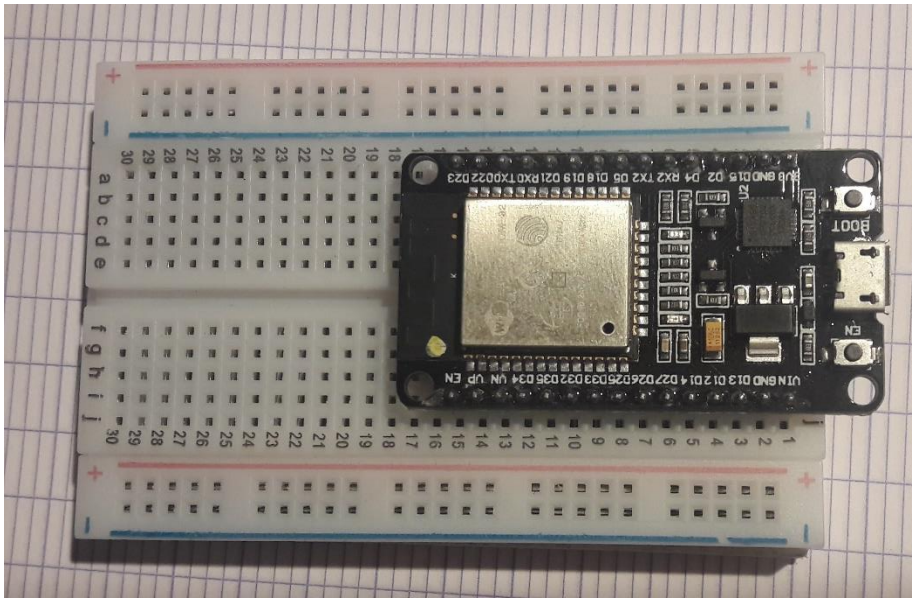
Le lab2 nous permet de nous familiariser avec l'ESP32 ainsi que le BME280.

Enfin, le projet consiste à utiliser le BME280 avec l'ESP32 afin de collecter des données (températures, pression atmosphérique) et de les envoyer en ligne sur un serveur.

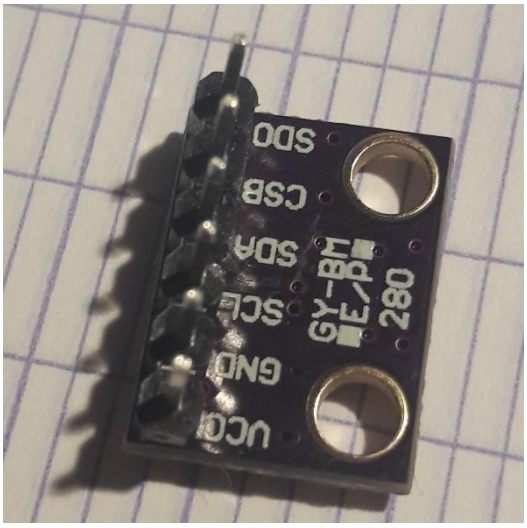
Marc NG KON TIA
Kajanan KANDAVEL

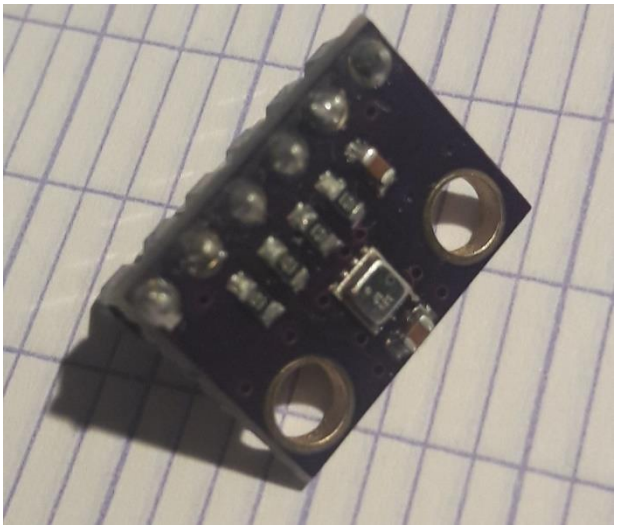
Présentation des équipements utilisés pendant les séances de TP :

La carte ESP32 : c'est un microcontrôleur doté de caractéristiques très intéressantes, puis qu'il possède un processeur de 160 ou 240 MHz, une mémoire de 520 KiB SRAM, et surtout en termes de connectivité gère le Wi-Fi ainsi que le Bluetooth. Le microcontrôleur possède également une trentaine de GPIO. Tous les aspects présentés précédemment font de l'ESP32 une bonne plateforme de développement pour travailler sur des projets de l'IoT.



Le capteur BME280 : c'est un senseur fabriqué par Bosch, nous permettant de mesurer la température, la pression atmosphérique ainsi que l'humidité. Il est idéal pour fabriquer des petites stations météos. L'intérêt de son utilisation ici est qu'on peut le brancher en I2C ou bien en SPI.



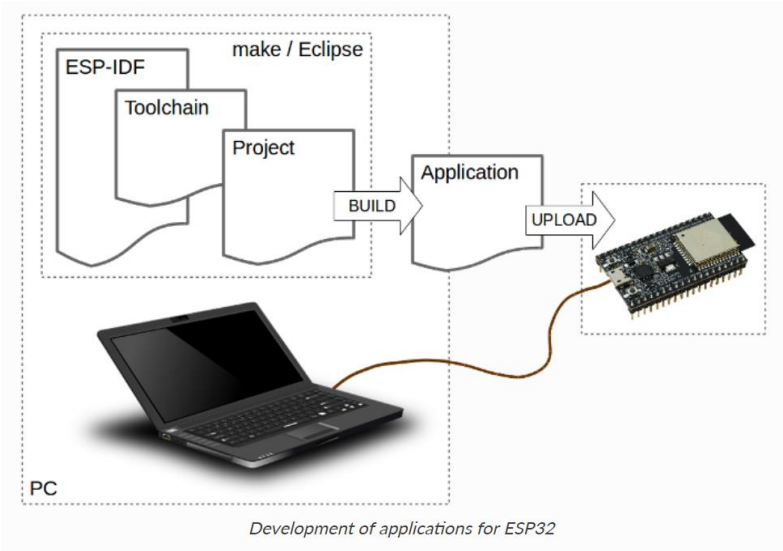


Marc NG KON TIA
Kajanan KANDAVEL

Installation du kit de développement :

La 1ère étape à effectuer est d'installer la toolchain correspondant à l'ESP32, afin de pouvoir développer sur notre ESP32. Ce fut pour nous l'un des principaux défis de ce projet, car sur tous les projets que nous avons eus depuis nos débuts à l'EFREI, nous développions sur un environnement Windows. Ici nous avons dû travailler avec un environnement Linux, et nous adapter à ses spécificités. Nous avons donc travaillé sous Ubuntu à travers une machine virtuelle. Cette partie là nous a causé une grande perte de temps, étant donné que nous n'étions pas du tout familiarisés avec l'environnement linux et que nous avons eu des soucis avec la VM sur les ordinateurs de l'école et que nous avons dû réinstaller l'environnement de développement une seconde fois sur nos propres ordinateurs…

Pour en revenir à notre sujet principal, l'intérêt de la toolchain est de pouvoir compiler le code d'un programme et de flasher notre carte ESP32 avec le programme compilé. Le schéma ci-dessous nous montre comment se déroule le développement d'application pour un ESP32.



Development of applications for ESP32

Pour expliquer un peu plus en détail le processus ci-dessus, en voici les 4 grosses étapes :

-Configuration d'un projet et écriture du code

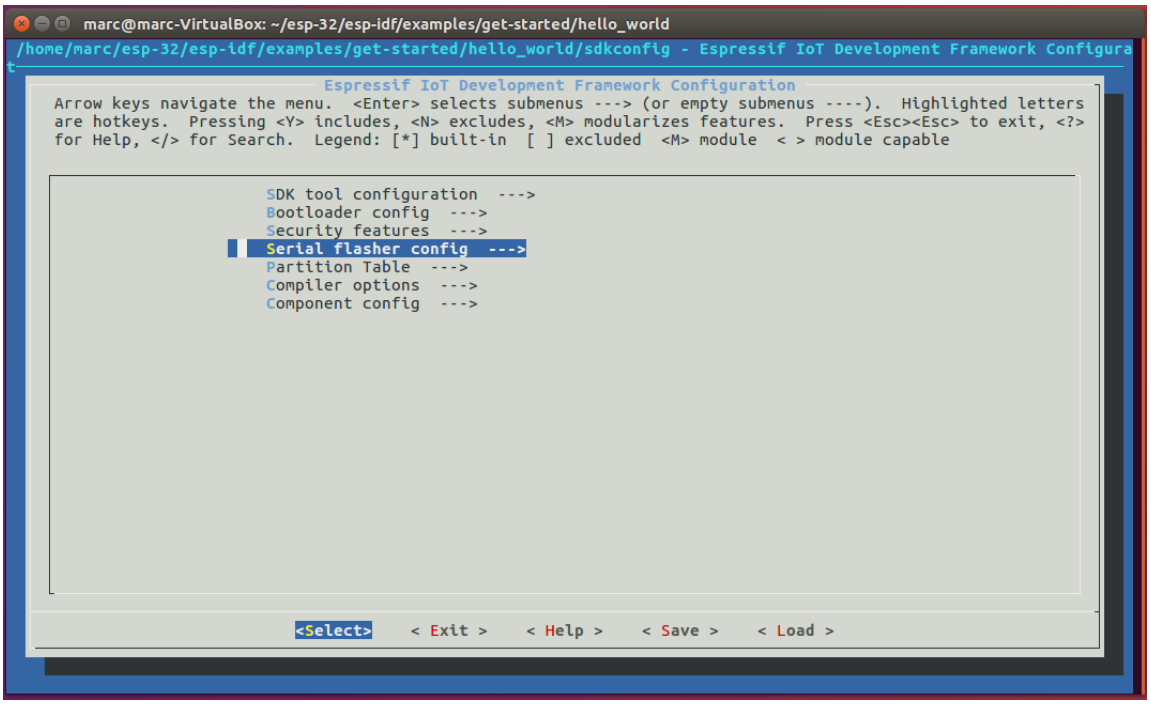
-Compilation du projet qui devient une application

-Flash(=chargement) de l'application sur l'ESP32

-Affichage/Déboggage de l'application

Nous avons donc installé le framework ESP-IDF sur notre VM en suivant les indications du Lab1. En important ESP-IDF nous avons également importé quelques programmes de test à flasher sur notre ESP32. Nous avons compilé le programme hello_world et nous l'avons flashé sur notre ESP32.

Marc NG KON TIA
Kajanan KANDAVEL




```
marc@marc-VirtualBox:~/esp-32/esp-idf/examples/get-started/hello_world$ make flash
WARNING: Toolchain version is not supported: 1.22.0-61-gab8375a
Expected to see version: 1.22.0-75-gbaf03c2
Please check ESP-IDF setup instructions and update the toolchain, or proceed at your own risk.
WARNING: Toolchain version is not supported: 1.22.0-61-gab8375a
Expected to see version: 1.22.0-75-gbaf03c2
Please check ESP-IDF setup instructions and update the toolchain, or proceed at your own risk.
Flashing binaries to serial port /dev/ttyUSB0 (app at offset 0x10000)...
esptool.py v2.1
Connecting........_
Chip is ESP32D0WDQ6 (revision 1)
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 19376 bytes to 11448...
Wrote 19376 bytes (11448 compressed) at 0x00001000 in 1.0 seconds (effective 150.4 kbit/s)...
Hash of data verified.
Compressed 136448 bytes to 65777...
Wrote 136448 bytes (65777 compressed) at 0x00010000 in 6.2 seconds (effective 176.5 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 82...
Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 1067.8 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting...
```

Pour checker que notre application est bien en train de tourner sur notre ESP32, nous utilisons la commande **make monitor.**

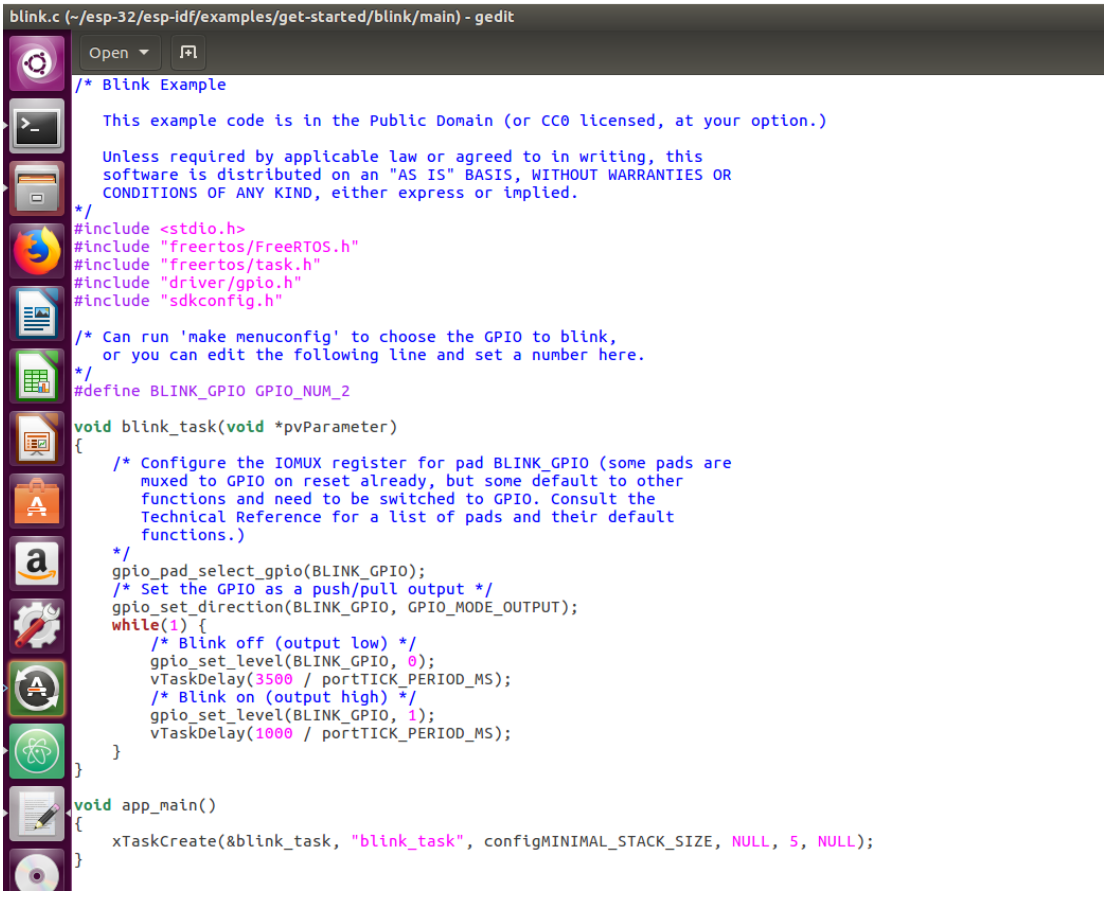
```
Hello world!
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 1, 4MB external flash
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
Restarting in 6 seconds...
Restarting in 5 seconds...
Restarting in 4 seconds...
Restarting in 3 seconds...
Restarting in 2 seconds...
Restarting in 1 seconds...
Restarting in 0 seconds...
Restarting now.
ets Jun  8 2016 00:22:57
```

Marc NG KON TIA
Kajanan KANDAVEL

Ceci est notre conclusion de la partie correspont au lab1.

En ce qui concerne le lab2, nous avons commencé à développer sur l'ESP32, c'est-à-dire que nous avons développé des applications ayant pour but de le faire réagir. Nous avons commencé avec un programme nommé **blink**, qui comme son nom l'indique, fait clignoter la LED de l'ESP32 lorsqu'on flash ce dernier. Le but de cet exercice est de nous faire découvrir le principe de fonctionnement des GPIO de l'ESP32. Les GPIO sont les ports d'entrée/sortie de nôtre microcontrôleur, ils permettent à la carte de communiquer avec d'autres circuits électroniques. C'est ce qui nous permettra notamment de le connecter au senseur BME280. La connexion se fait d'ailleurs via le bus SPI.

Comme vous pouvez le voir sur le code du programme, le GPIO qui correspond à la LED de l'ESP32 est le GPIO numéro 2. On choisit ensuite la fréquence de clignotement de la LED, qu'on peut faire varier dans les vTaskDelay. Celle-ci est indiquée en milliseconde.

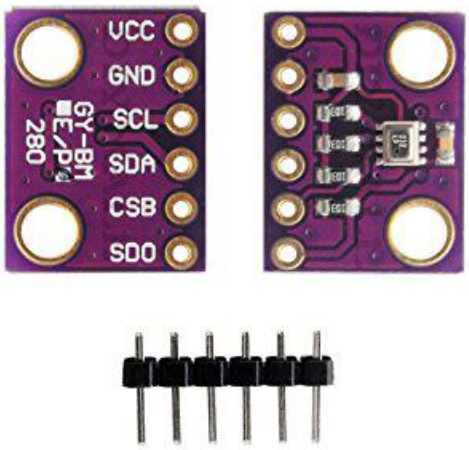



```c
/* Blink Example

   This example code is in the Public Domain (or CC0 licensed, at your option.)

   Unless required by applicable law or agreed to in writing, this
   software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
   CONDITIONS OF ANY KIND, either express or implied.
*/
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "sdkconfig.h"

/* Can run 'make menuconfig' to choose the GPIO to blink,
   or you can edit the following line and set a number here.
*/
#define BLINK_GPIO GPIO_NUM_2

void blink_task(void *pvParameter)
{
    /* Configure the IOMUX register for pad BLINK_GPIO (some pads are
       muxed to GPIO on reset already, but some default to other
       functions and need to be switched to GPIO. Consult the
       Technical Reference for a list of pads and their default
       functions.)
    */
    gpio_pad_select_gpio(BLINK_GPIO);
    /* Set the GPIO as a push/pull output */
    gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);
    while(1) {
        /* Blink off (output low) */
        gpio_set_level(BLINK_GPIO, 0);
        vTaskDelay(3500 / portTICK_PERIOD_MS);
        /* Blink on (output high) */
        gpio_set_level(BLINK_GPIO, 1);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
}

void app_main()
{
    xTaskCreate(&blink_task, "blink_task", configMINIMAL_STACK_SIZE, NULL, 5, NULL);
}
```

Marc NG KON TIA
Kajanan KANDAVEL

Pour ce qui est du projet enfin, le principe est simple : relier le senseur BME280 à la carte ESP32, récupérer les informations (Température, Pression Atmosphérique) du capteur BME280 avec notre ESP32 et les envoyer sur un serveur pour les stocker dans un fichier au format JSON.

La connexion entre l'ESP32 et le BME280 se fait donc via le bus SPI. Notre capteur possède différents ports :



VCC et GND sont pour l'alimentation.

SCL est un port d'entrée correspondant au signal d'horloge.

SDA est un port d'entrée de donnée, pour les données envoyées par notre ESP32 à notre BME280.

SDI est un port de sortie de donnée, pour les données envoyées par notre BME280 à notre ESP32.

Enfin CSB correspond au chip select.

Monsieur Massinissa nous a fourni une architecture de code correspondant à ce projet, que nous devions compléter. Le projet était évidemment divisé en plusieurs fichiers de code :

Sensor_reading.c, qui nous permet d'effectuer la connexion entre notre ESP32 et BME280, et de récupérer les données du BME280. Les données du BME280 sont stockées dans une structure créée par nos soins, avec un champ correspondant à la donnée et l'autre à la date à laquelle nous l'avons récupérée. Chaque mesure récupérée dans cette structure est ensuite stockée dans un tableau appelé transmission queue.

Transmission.c et formatting.c sont ensuite utilisés pour récupérer nos mesures stockées dans un tableau, et les convertir au format JSON. Chaque objet JSON doit être converti en string pour pouvoir être transféré sur un serveur via le protocole HTTP.

On fait appel à App_layer.c pour transférer nos données via une requête HTTP.

Trans_layer.c se charge lui de d'envoyer les données au serveur.

Le fichier connectivity.c nous permet de gérer la connexion au serveur. Pour ce qui est de cette connexion, elle se fait via un hotsport WiFi activé par nos soins.

Marc NG KON TIA
Kajanan KANDAVEL

Conclusion :

Nous n'avons malheureusement pas pu aller plus loin sur ce projet, et faire progresser la base fournie par notre professeur, nous avons eu énormément de difficultés au niveau technique afin d'accomplir ce projet. C'est notre plus grand regret, car ce module d'IoT était vraiment rafraîchissant, un peu éloigné du thème de notre majeure mais pas tant que ça finalement, étant donné les possibilités que l'IoT offre avec le Cloud. Nous étions particulièrement motivés, surtout après les cours magistraux qui ont permis de nous offrir une première vision de l'IoT. Mais nous pensons justement que ce cours doit être une introduction à l'IoT et non une découverte complète de l'IoT. C'est pourquoi nous avons quelques regrets sur le sujet de projet qui était très spécifique et demandait une connaissance approfondie du développement lié à l'IoT(que ce soit au niveau de l'ESP32 ou du BME280), même avec l'aide de la documentation.

Pour ce qui est de la répartition du travail, nous avons toujours raisonné en binôme sur les différents TP/projet, afin que nous puissions comprendre tous les 2. Pour ce qui est des manipulations avec le matériel spécifique c'est plus Marc qui a été actif sur ce plan-là, sur la partie code également.