Rapport de Synthèse

N°75

Objet connecté à base de microcontrôleur ARM

Auteurs : CHEN Muyao & YE Jing

Encadrant : ABIB Idir Ghalid

Table des matières

[**Introduction** 3](#_Toc517016463)

[**Partie Hardware** 4](#_Toc517016464)

[**Matériel utilisé** 4](#_Toc517016465)

[**Explication des programmes** 4](#_Toc517016466)

[**“Project” en détail** 4](#_Toc517016467)

[**Fonction “Main.c”** 5](#_Toc517016468)

[**Gestion du protocole TCP** 7](#_Toc517016469)

[**Gestion des connexions Bluetooth** 10](#_Toc517016470)

[**Gestion de l’enregistrement de donnée** 12](#_Toc517016471)

[**Périphériques** 12](#_Toc517016472)

[**Partie Android** 12](#_Toc517016473)

[**Introduction** 12](#_Toc517016474)

[**IDE** 13](#_Toc517016475)

[**Explication des programmes** 13](#_Toc517016476)

[**Structure du programme** 13](#_Toc517016477)

[**Classe “TemhumDetection.java”** 14](#_Toc517016478)

[**Classe “TemperatureHistory.java”** 17](#_Toc517016479)

[**Classe “LightControl.java”** 18](#_Toc517016480)

[**Layout d’application** 19](#_Toc517016481)

**Conclusion**………………………………………………………..…………………………………19

# **Introduction**

Les applications embarquées et les objets connectés nécessitent souvent une connexion sans fil type WiFi ou Bluetooth afin d’interagir avec un système distant. Ceci permet de récupérer des données de capteurs via une connexion directe ou via des points d’accès.

L’enjeu principal de l’Internet des objets de nos jours est l’interopérabilité entre différents protocoles de communication. Ce projet est dédié au développement d’un point d’accès pour gérer à la fois la liaison Internet (WiFi) et la communication avec les capteurs (Bluetooth), qui permettra aux utilisateurs de facilement contrôler leurs objets connectés via une seule application (Android).

Pour des raisons de faible coût et de faible consommation électrique, certains systèmes embarqués intègrent un microcontrôleur de type ARM et disposent même d’un contrôleur WiFi Intégré. Vu la mémoire (Programme et données) disponible dans un microcontrôleur, l’optimisation du programme principal et des bibliothèques TCP/IP (Ethernet) ou Bluetooth est cruciale. En outre, le développement de l’application Android nécessite une bonne connaissance de la programmation orientée objet, et aussi un savoir-faire du génie logiciel.

Ce projet est divisé en deux parties. Les objectifs de la partie dite "Hardware" sont d’une part, la mise en place d’un point d’accès (Gateway) WiFi à l’aide d’un microcontrôleur ARM Cortex M4 de Texas Instruments permettant la communication avec un smartphone; et d’autre part, l’ajout à ce même microcontrôleur d’une liaison Bluetooth afin de communiquer avec des objets connectés tels que des capteurs. Pour la partie "logicielle", une application Android sera développée pour établir la communication entre le smartphone et le point d’accès via WiFi. Cette application a pour but de recevoir les données, de les visualiser et de contrôler des objets tel qu’un interrupteur de lumière.

# **Partie Hardware**

## **Matériel utilisé**

Afin de créer une passerelle IoT qui peut à la fois gérer plusieurs connexions en Bluetooth et la connexion Ethernet, nous avons choisi un microcontrôleur ARM cortex M4 (TIVA C TM4C1294) proposé par Texas Instruments.

Par rapport aux périphériques, nous avons utilisé deux cartes Arduino UNO, une carte Wemos D1 (ESP8266 Wi-Fi intégré) et deux modules DHT11 (détection)

L’IDE utilisé pour le développement en C est CCS7.3.0 (Code Composer Studio) de TI.

## **Explication des programmes**

### **“Project” en détail**

Le projet CCS comprend des fichiers nécessaires pour programmer les fonctionnalités sur la carte. D’abord, il existe trois répertoires qui s’appellent “Binaires”, “Includes” et “Debug”. Ces deux répertoires n’existent qu’après avoir construit (Build) le projet. Dans le répertoire “Binaires”, on trouve le fichier généré (compilé et construit) par l’IDE pour enfin programmer le microcontrôleur. Le répertoire “Includes” contient toutes les librairies utilisées par nos programmes. Dans le répertoire “Debug”, il y a tous les fichiers intermédiaires, y compris le Makefile.

Ensuite, nous trouvons tous les fichiers pour construire notre projet. Dans “drivers”, le fichier “pinout.c” initie tous les ports GPIO par déclarer leurs types et aussi les configure selon nos besoins. Dans “utils”, le fichier “ustdlib.c” offre des fonctions pratiques et standardise afin de faciliter la programmation. Le fichier “uartstdio.c” nous fournit les fonctions utilisant UART0 pour envoyer les textes qu’on veut afficher sur l’écran d’ordinateur par la liaison série. Les deux autres fichiers C concernent la gestion de LwIP. La partie TCP sera expliquée par la suite.

Puis, “enet\_lwip.c” contient la fonction “main” du programme et aussi quelques sous-fonctions. Le fichier “tcp\_server.c” et “tcp\_server.h” sont des fonctions pour la gestion du protocole TCP. Le fichier “lwipopts.h” définit les options pour utiliser la pile LwIP.  Le fichier “startup\_css.c” est pour démarrer (bootloader) la carte pour qu’il puisse charger la fonction “main” et ainsi les autres fichiers dépendants. Enfin, les fichiers restants sont nécessaires pour le compilateur.

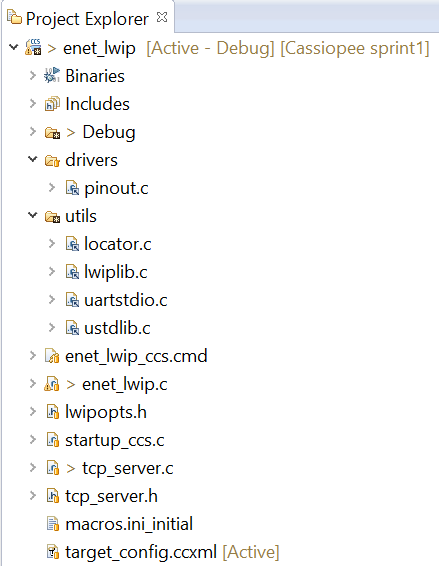


Fig 1 Fichiers du projet

### **Fonction “Main.c”**

Nous aborderons dans cette partie le principe de la fonction “main”. L’explication détaillée du code est commentée dans le programme.

La fonction “Main” contient trois étapes. La première phase et celle d’initialisation, où on initialise la fréquence de microcontrôleur, la configuration des broches, les liaisons UART, et l’EEPROM.

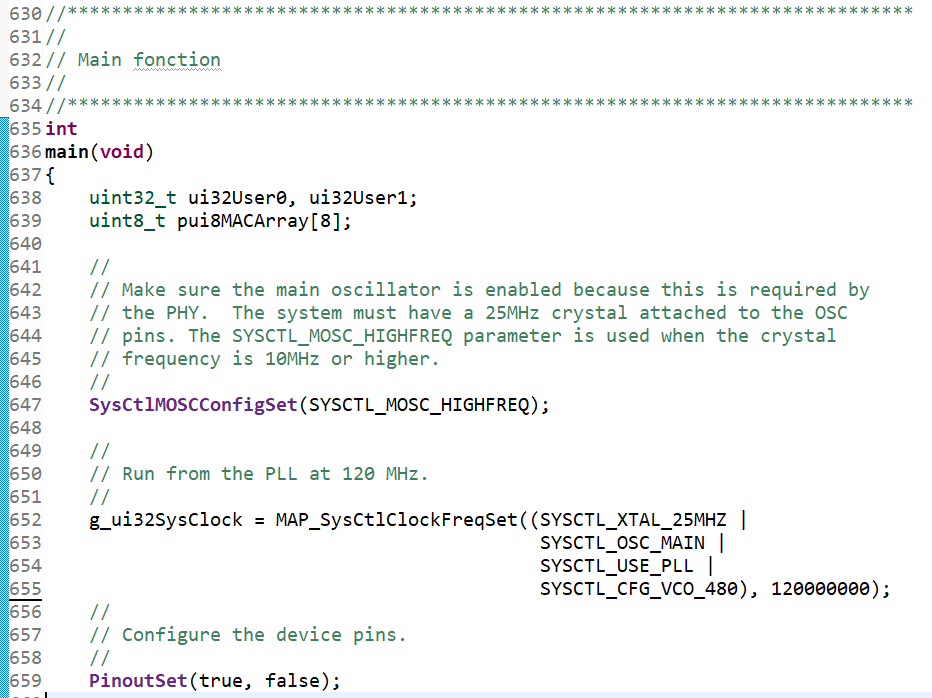


Fig 2 Initialisation

La deuxième phase concerne les premières manipulations essentielles. On configure le module Ethernet intégré sur la carte pour d’abord se connecter à Ethernet et obtenir une adresse IP locale et ensuite mettre en place un serveur TCP à l’aide des fonctions spécifiques dont je parlerais dans la partie suivante. En plus, on ajoute les adresses des deux appareils Bluetooth dans la liste des capteurs Bluetooth pour les futures interrogations.

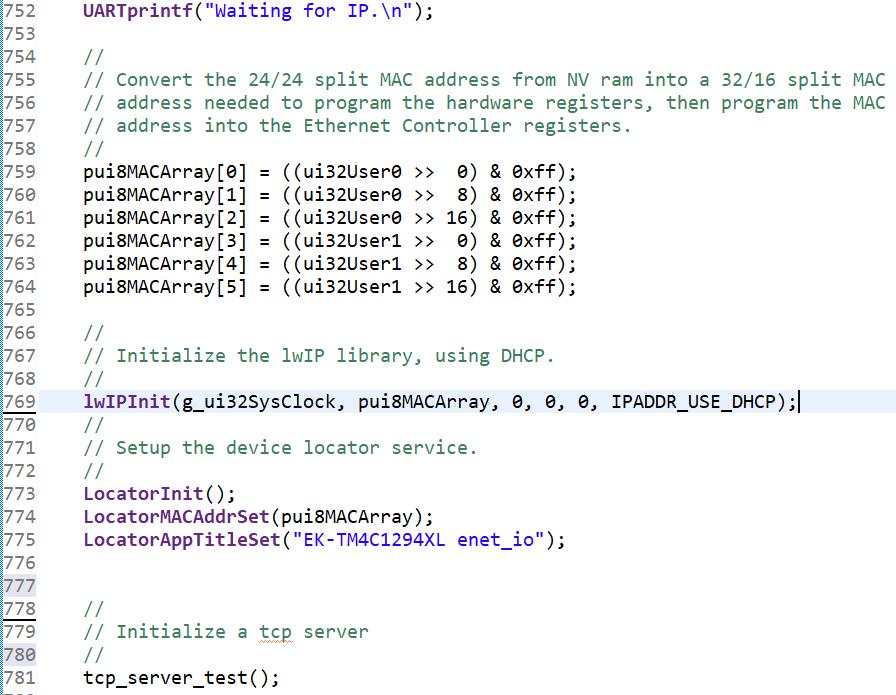


Fig 3 Connexion Ethernet

Dans la boucle infinie (while(1)), on essaie de se connecter avec chaque capteur Bluetooth en changeant le pair du module Bluetooth lié à la carte maître. En envoyant un message de demande, on récupère ensuite dans l'interruption d’URAT7 la donnée renvoyée par le capteur connecté. Toute la communication sur Internet sera traitée dans l’interruption d’Ethernet.

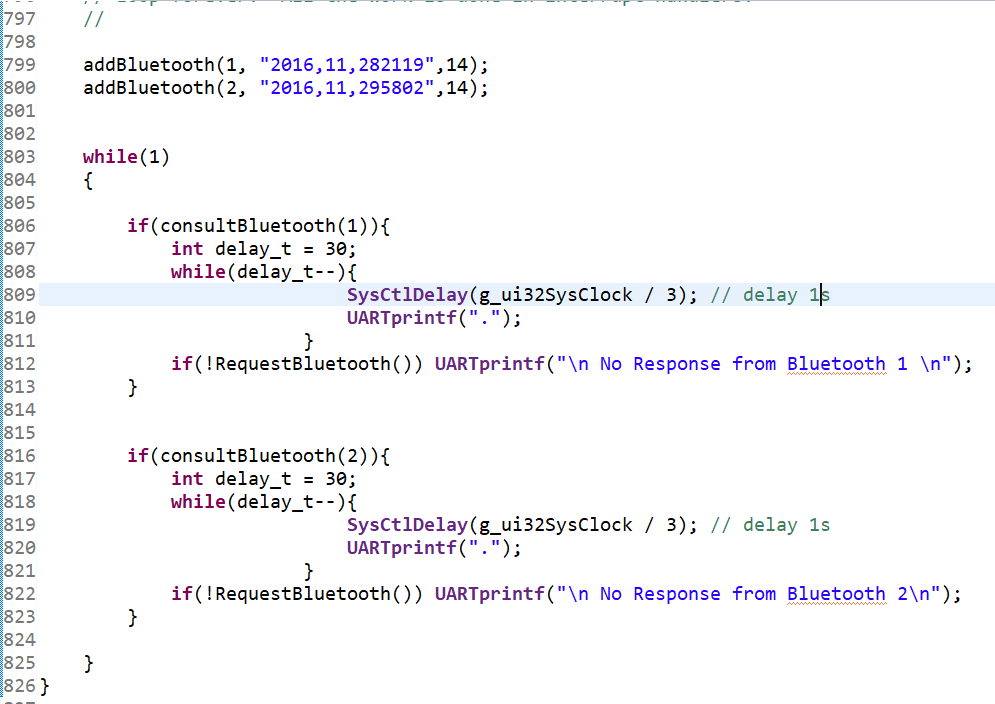


Fig 4 Boucle infinie

### **Gestion du protocole TCP**

Il existe plusieurs manières pour mettre en place la communication en protocole TCP/IP, y compris les sockets, LwIP (Light-Weight IP), et uIP(Micro IP). La programmation en sockets est facile à manipuler, mais elle demande un système d’exploitation (RTOS). Vu que notre programme n’est pas trop complexe, et qu’en plus, nous voulons réduire la consommation du système embarqué, nous avons donc choisi d’intégrer LwIP, une pile de protocole TCP dédiée aux systèmes embarqués.

La communication Ethernet s’établit en deux phases, celle d’initialisation et celle de traitement. Dans l’initialisation, on crée d’abord un identifiant (sous forme d’une structure) pour s’identifier dans le réseau local. Au niveau du réseau, on définit principalement notre adresse MAC et la manière d’adressage, soit par le protocole DHCP, ou par une adresse IP statique.

Après l’initialisation, on arrive à la phase critique du projet, où on créé un serveur TCP pour échanger de données. Afin d’établir un serveur, il faut d’abord créer un tcp\_pcb (protocol control block) pour mettre la configuration du serveur (adresse IP locale, Port local, adresse IP du client, Port du client, etc.), et demander une allocation de mémoire qu’on va utiliser comme un buffer de données à envoyer et à recevoir. En utilisant la fonction “tcp\_bind()”, on peut relier notre PCB avec un port spécifique. Ensuite, on accepte la connexion venant d’un client.

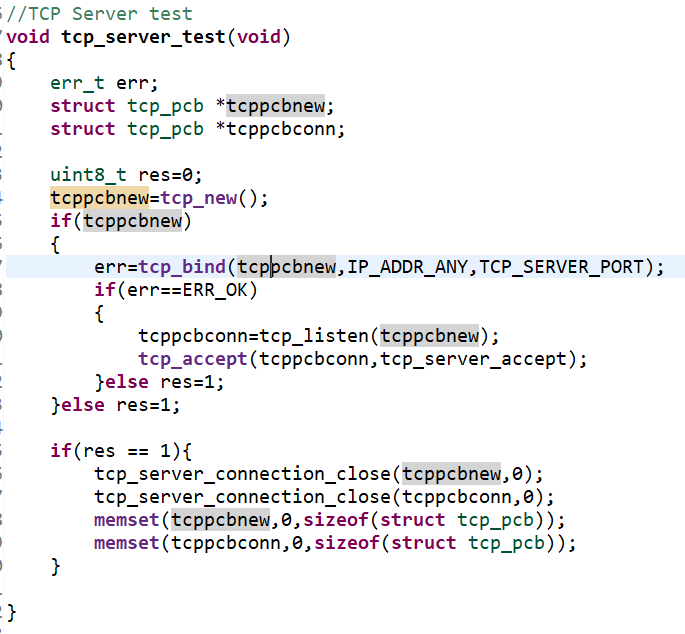


Fig 5 tcp\_server\_test

Les traitements qu’on effectue lorsqu’une demande de connexion arrive sont décrits dans la fonction “tcp\_server\_accept()”. En connexion, on va mettre à jour les informations du client et aussi l’état de connexion. Nous relions aussi les fonctions qu’on a créées pour manipuler les données avec les modèles d’handler tcp proposés par LwIP.

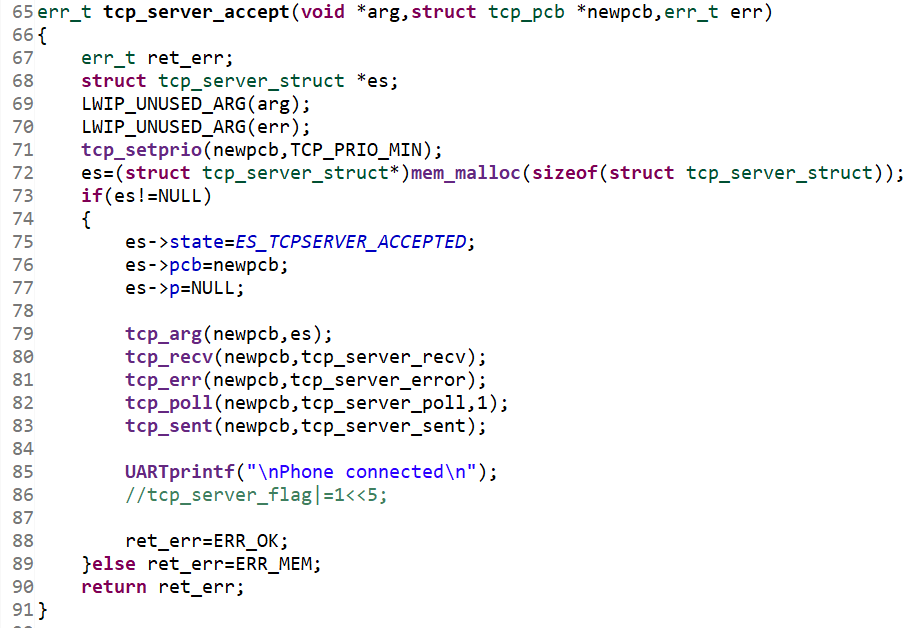


Fig 6 tcp\_server\_accept()

En recevant une donnée, nous allons la récupérer de la mémoire du paquet TCP et la mettre dans un buffer. On change l’état d’un drapeau signifiant qu’il y a une nouvelle donnée à traiter.

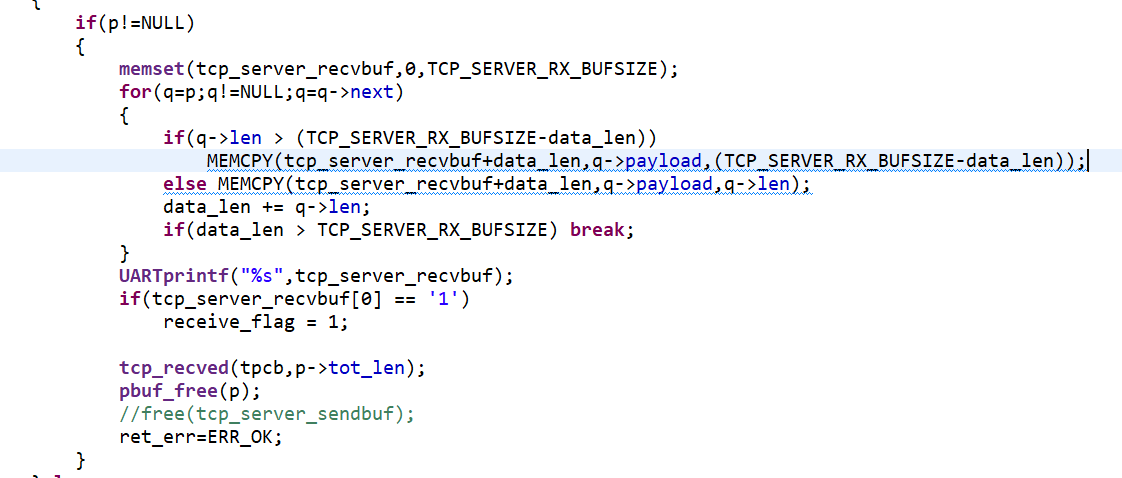


Fig 7 tcp\_server\_recv()

Dans une autre fonction “tcp\_server\_poll()”, nous testons le drapeau et s’il y a une nouvelle donnée, on va la comparer avec “1”. Si c’est “1”, que l’on a reçu, on va envoyer toutes les données enregistrées dans EEPROM et la vider. Enfin, on remet le drapeau à 0. Cette fonction va être appelée régulièrement (chaque 2ms) pour vérifier qu’on n’omet pas une demande.



Fig 8 tcp\_server\_poll

Pour envoyer des données, il faut juste mettre des données dans le packet TCP, et appliquer la fonction “tcp\_sndbuf()” fournie par LwIP.

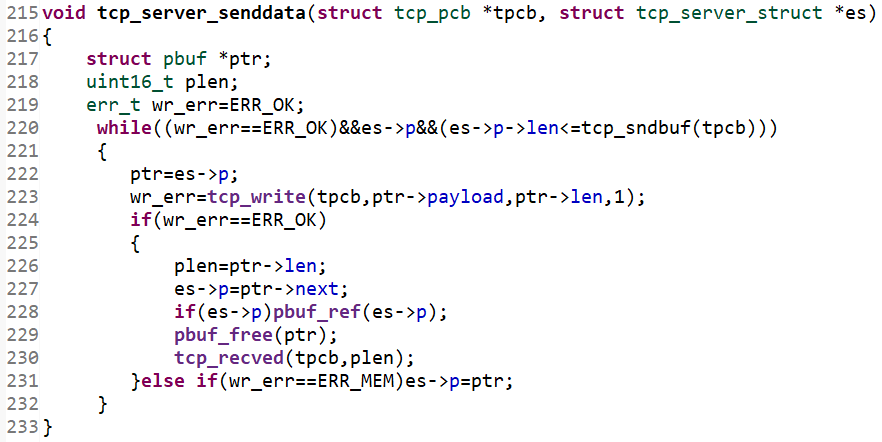


Fig 9 tcp\_server\_senddata

Finalement, on ferme la session quand le client se déconnecte et attendre une nouvelle connexion.

### **Gestion des connexions Bluetooth**

Comme nous l’avons expliqué, les demandes venant de l’application Android vont être traitées dans l’interruption Ethernet. La communication entre la passerelle et les capteurs Bluetooth se fait dans la boucle infinie dans la fonction “main”

Nous connectons un Module HC05 avec la carte TI, et configurons ce module par commandes AT via UART (liaison série). On change entre le mode AT et le monde transmission en contrôlant la broche “EN” sur module. Si on met “EN” à “1”, puis alimenter le module, il va entrer dans le mode AT. Si on désactive “EN” et redémarre le module, il va passer au mode communication. C’est de cette manière qu’on peut connecter ce module avec plusieurs capteurs Bluetooth.

Le capteur bluetooth s’identifie par une adresse Hardware unique. Ainsi, on sauvegarde les adresses du capteur dans un tableau.

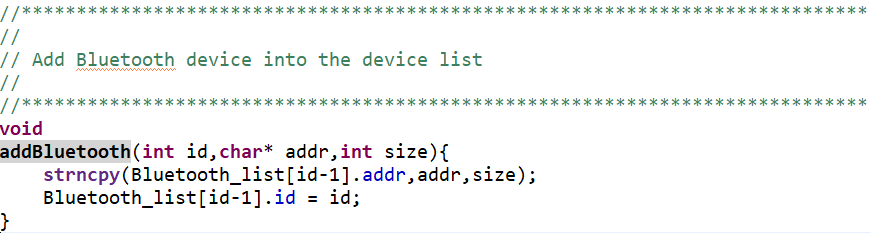


Fig 10 addBluetooth

Dans la fonction “consultBluetooth()”, on coupe la connexion Bluetooth actuelle et entre dans mode AT. Ensuite, on envoie des commandes pour redéfinir le capteur à connecter. Enfin, on redémarre le module pour se connecter avec ce capteur.

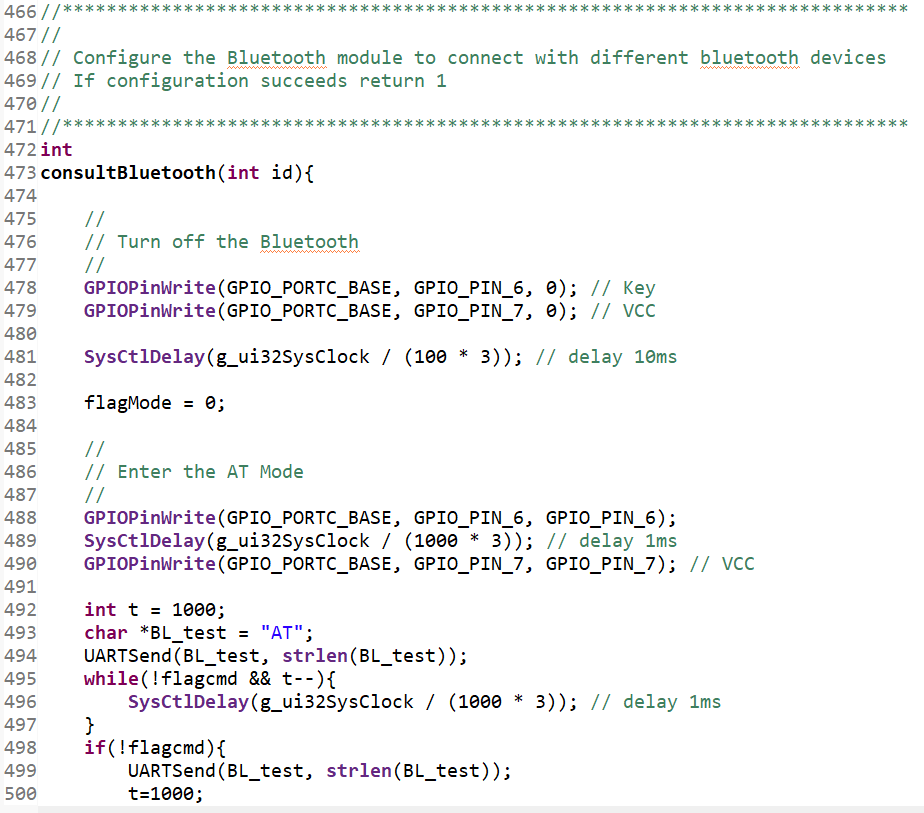


Fig 11 consultBluetooth

Finalement, dans la fonction “RequestBluetooth()”, on envoie une demande (“1”) à un capteur Bluetooth. On attend une seconde puis vérifier s’il y a une nouvelle donnée stockée dans EEPROM.

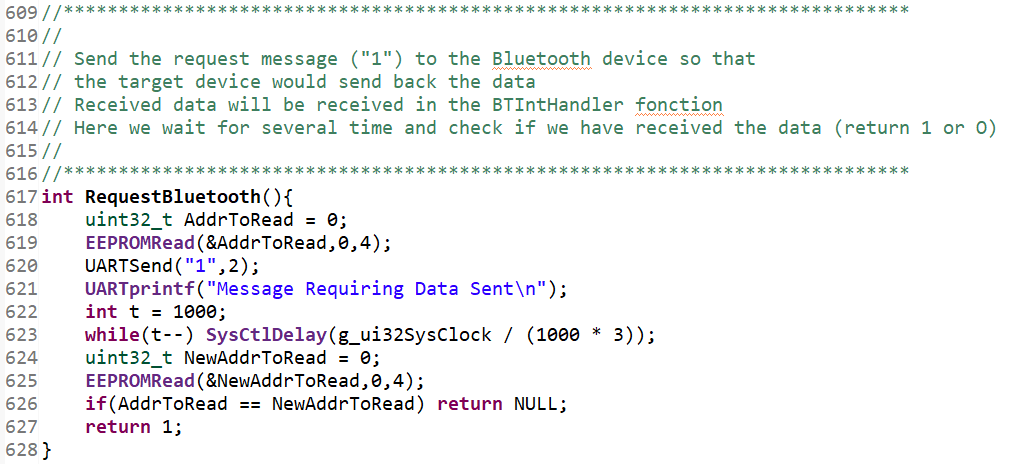


Fig 12 RequestBluetooth

### **Gestion de l’enregistrement de donnée**

Pour gérer les données récupérées des capteurs Bluetooth, nous avons décidé d’utiliser EEPROM. L’intérêt de cette mémoire est qu’on ne perd pas de données même si la carte n’est plus alimentée. Car l’espace de la mémoire est limité, on vide la mémoire quand on envoie tous les données à l’application Android.

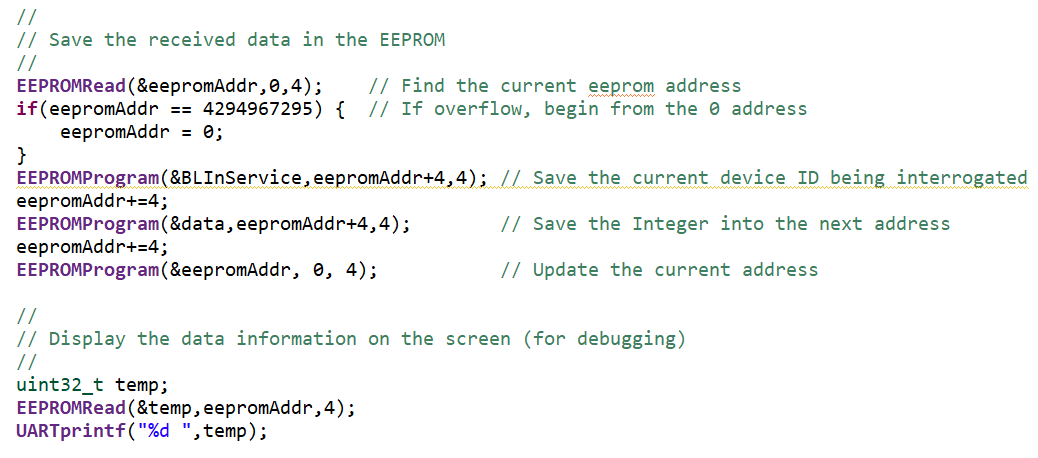


Fig 13 EEPROM

## **Périphériques**

Afin de démontrer notre système IoT, nous avons aussi programmé des périphériques, y compris deux capteurs Bluetooth et un actionneur. Nous avons utilisé deux cartes Arduino UNO et deux modules DHT11 pour détecter la température et l’humidité. Ces deux cartes sont aussi liées avec un module HC05. On a programmé une carte WEMOS D1 qui intègre un module ESP8266 pour illustrer que notre application arrive aussi à contrôler un actionneur via la liaison Wi-Fi. Une LED est donc liée avec cette carte.

# **Partie Android**

## **Introduction**

Cette application sert à réaliser la détection de l’environnement intérieur et le contrôle des lumières en utilisant la communication par les sockets avec un serveur TCP.

## **IDE**

Les ressources logicielles sont :

Android Studio 3.0.1, com.github.markushi:circlebutton:1.1,

com.github.jd-alexander:android-flat-button:v1.1,

com.github.zcweng:switch-button:0.0.3@aar,  CSVWriter, hellocharts-library-1.5.8

## **Explication des programmes**

### **Structure du programme**

Notre application dispose de 2 fonctions : lecture de la température et de l’humidité; et le contrôle de lumière. Le fichier MainActivity.java correspond à la page principale de l’application. Les deux fonctions ont besoin la connexion TCP pour communiquer avec la partie Hardware grâce à tcpClientSender.

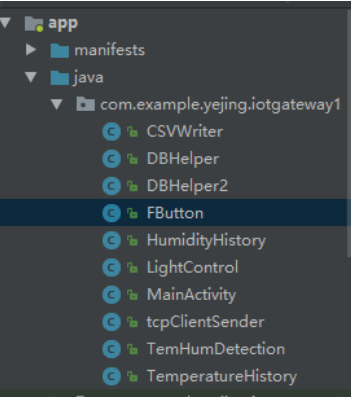
                        

Fig 13 Page principale

La fonction “Détection de la température et de l’humidité” est réalisée par TemhumDetection.java. Il faut saisir l’adresse IP de la carte à laquelle on se connecte pour commencer la communication. Dès que c’est fait, le bouton “Synchronize” permet d’obtenir les données de la température et de l’humidité stockées dans la carte en envoyant un “1”. Elles sont sauvegardées dans les bases de données correspondantes grâce à la classe DBHelper. Les deux boutons “Temperature” et “Humidity” sont respectivement liés avec TempratureHistory et HumidityHistory, par lesquelles on peut voir les données sous forme d’un graphe (20 dernières données) avec l’instant où la valeur est détectée et d’un tableau qui présente toutes les données. Elles peuvent être exportées en fichier .CSV en utilisant CSVWriter.

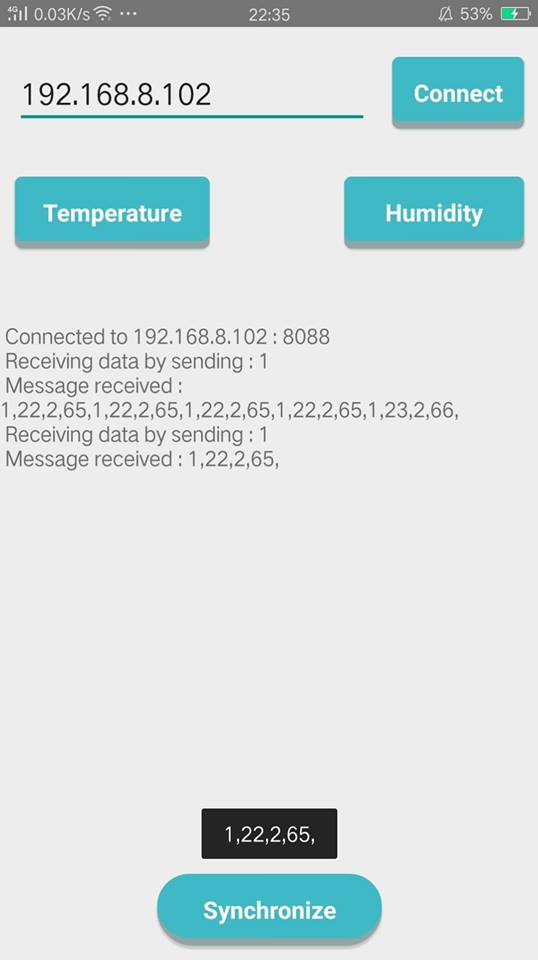
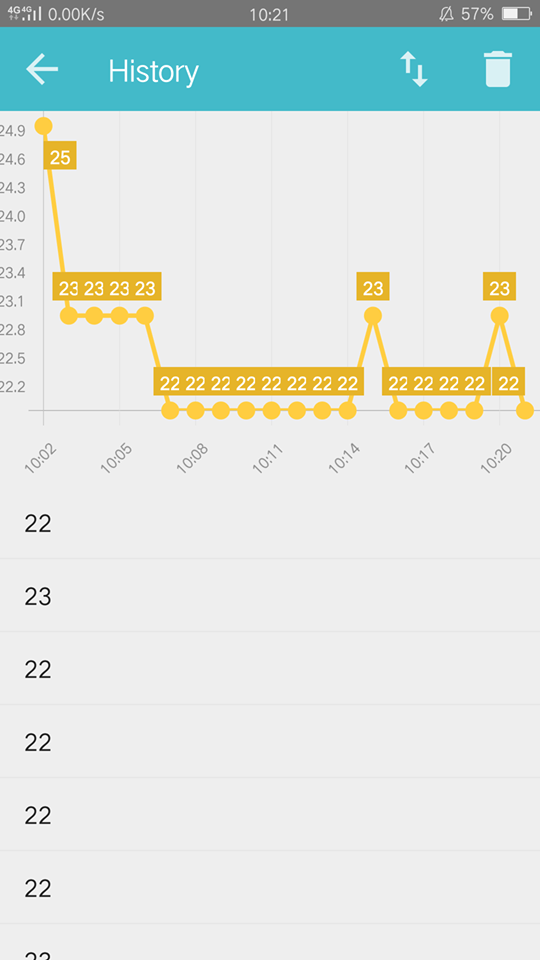
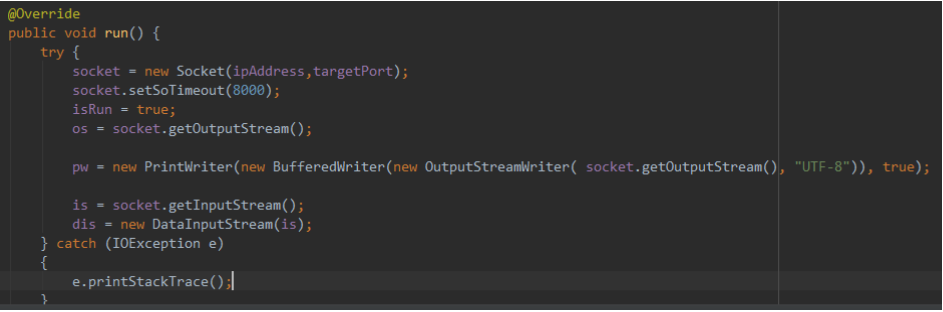
                              

Fig 14 Détection de la température et de l’humidité

La deuxième fonction est beaucoup plus simple, elle comporte un bouton permettant d’allumer et d’éteindre la lumière. C’est la classe LightControl qui s’occupe de cette fonction.

### **Classe “TemhumDetection.java”**

Dans TemhumDetection, on réalise la communication TCP avec la carte qui est comme un serveur et envoie des messages de notre choix. Le processus entier est présenté en bas de l’écran par la méthode « append ». Toutes les étapes de connexion sont encapsulées dans TCPClientSender (setAddress, send, receive, quit, etc) pour gérer la connexion dans un autre Thread et pour simplifier le code. Il utilise un PrintWriter et la méthode getOutputStream pour stocker le message à envoyer au serveur et la méthode getInputStream pour stocker le message reçu de serveur.

 Fig 15 Déroulement de TCP Client Sender

Au début, on fixe le numéro de port et on tape l’adresse IP à laquelle on veut se connecter pour créer un nouveau socket. Ensuite, on exécute un TCPClientSender(mTCPCL) afin de commencer la connexion. A ce moment-là, le paramètre « isRun » de mTCPCL est égal à « true ». Quand un socket est créé, bind est réalisé automatiquement. On peut dès maintenant échanger des données avec le serveur.

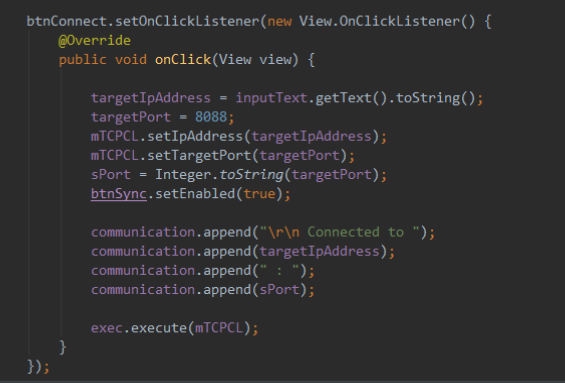


Fig 16 Conexion TCP

On clique le bouton « Synchronize » pour envoyer un “1” en appelant la méthode « send » dans TCPClientSender.

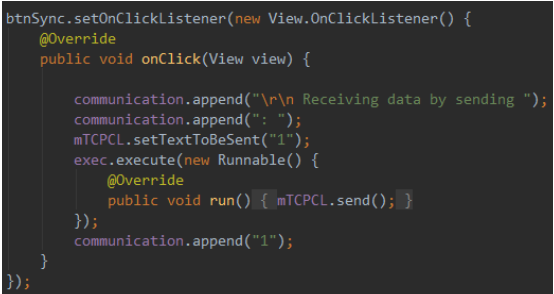


Fig 17 Demande de données

Quand le serveur nous donne une réponse, TCPClientSender la garde et l’associe à une action « tcpClientReceiver ». Et puis TemhumDetection peut détecter cette action et appeler « myHandler » pour traiter les données et afficher le message sur l’écran.

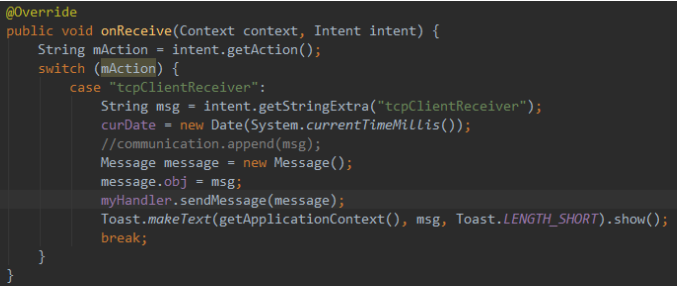


Fig 18 Réception de données

Dans MyHandler, les données sont triées selon le chiffre précédant la valeur de température ou d’humidité. Si c’est un “1”, la valeur est stockée dans la base de données “Temperature.db”; si c’est un “2”, la valeur est stockée dans “Humidity.db”.



Fig 19 Traitement de données

### **Classe “TemperatureHistory.java”**

Cette classe est appelée quand le bouton “Temperature” est cliqué. Elle se compose de 3 parties. D’abord le toolbar en haut permet d’afficher le titre et les boutons pour exporter les données en fichier .csv et vider la base de données.

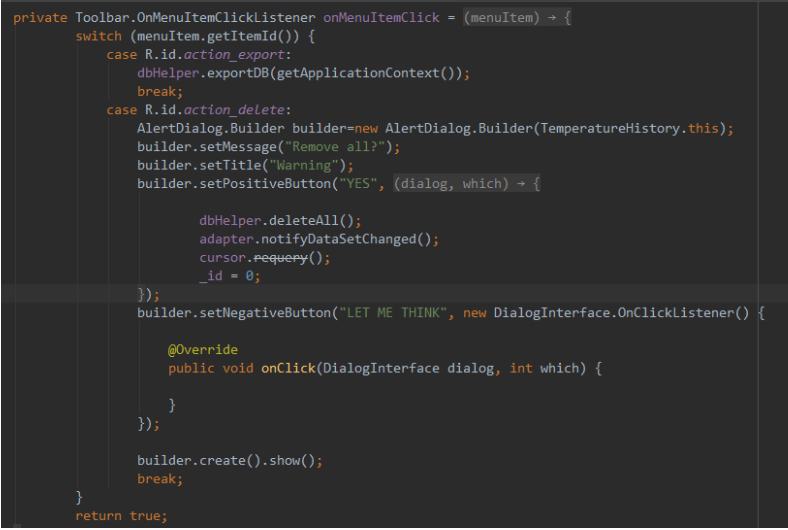


Fig 20 Toolbar

Ensuite, c’est un graphe qui affiche les 20 dernières valeurs reçues. Elles sont trouvées par un curseur de DBHelper et sauvegardées dans une table de 20 éléments. Les fonctions pour dessiner le diagramme sont fournies par la librairie “hellocharts”. Au-dessous c’est un tableau avec toutes les données reçues. S’il n’existe pas de données, cette application affiche “Nothing received yet” et la page est blanche.



Fig 21 Affichage de diagramme

### **Classe “LightControl.java”**

Cette classe suit presque le même principe que la classe “TemhumDetection”, sauf qu’ici l’adresse IP est fixe. Au milieu de la page apparait un grand bouton de commutation.  Quand on le glisse à gauche, il envoie un “1” à la lumière pour l’allumer. Et un “0” est envoyé en glissant le bouton à droite pour éteindre la lumière.

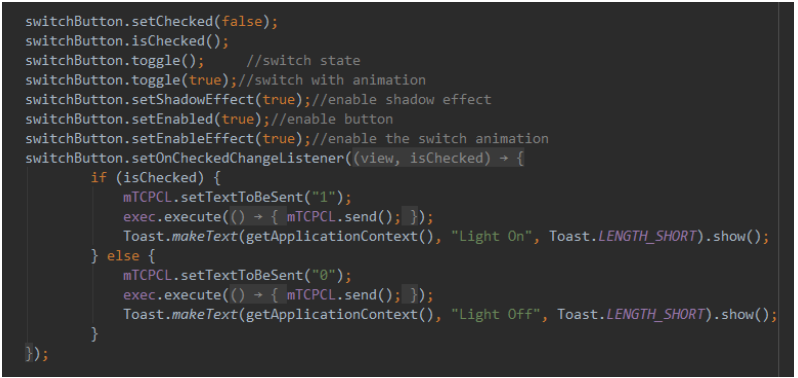


Fig 22 Switch Button

### **Layout d’application**

Les fichiers .xml se trouvant à java/res/layout gèrent l’interface graphique de l’application Android. Dans ces fichiers on peut préciser la disposition globale des pages, les éléments utilisés et leur taille, position, couleur, contenu, etc. La façon de lier ces fichiers avec les fonctions de java est d’appeler “setContentView(R.layout.xxx)” ou “findViewById(R.id.xxx)” quand on initialise les éléments par code java selon les cas différents.

## **Conclusion**

Ce projet ambitieux a mêlé deux aspects : du hardware et du software pour l’embarqué. Nous avons programmé en C un microcontrôleur ARM Cortex M4 de TI en utilisant une librairie LwIP pour gérer le protocole TCP/IP sur un système à faible ressources et à basse consommation. Ce microcontrôleur interagit avec différents capteurs en Bluetooth. Nous avons également développé une application Android mêlant programmation socket, base de données et affichage de graphe.

Ainsi, ce projet nous a permis de nous familiariser un peu plus avec les systèmes embarqués. La figure suivante montre notre système autour de la carte Tiva C mettant en œuvre le microcontrôleur ARM Cortex M4 de TI.

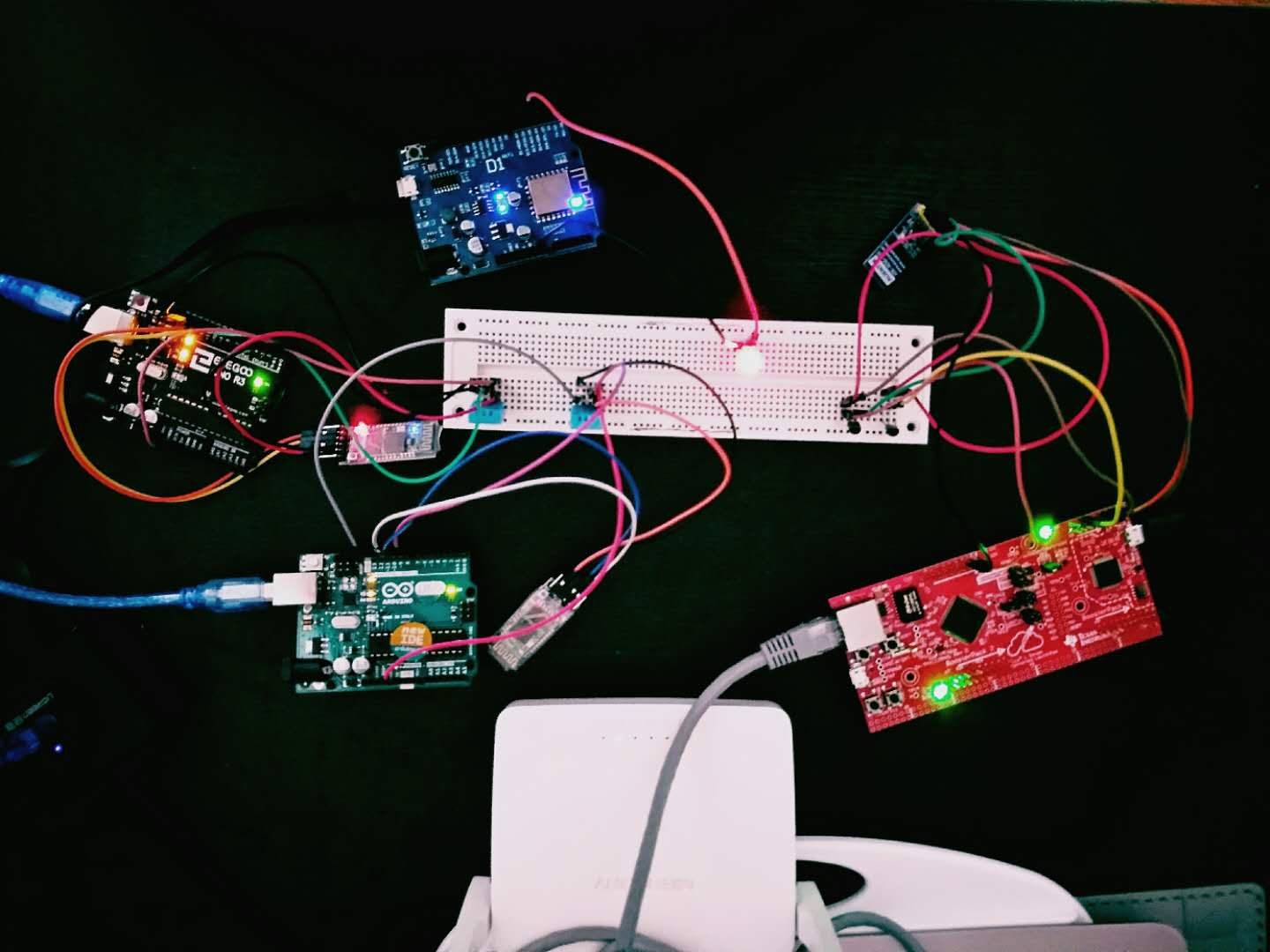


Fig 23 Objet connecté à base de microcontrôleur ARM