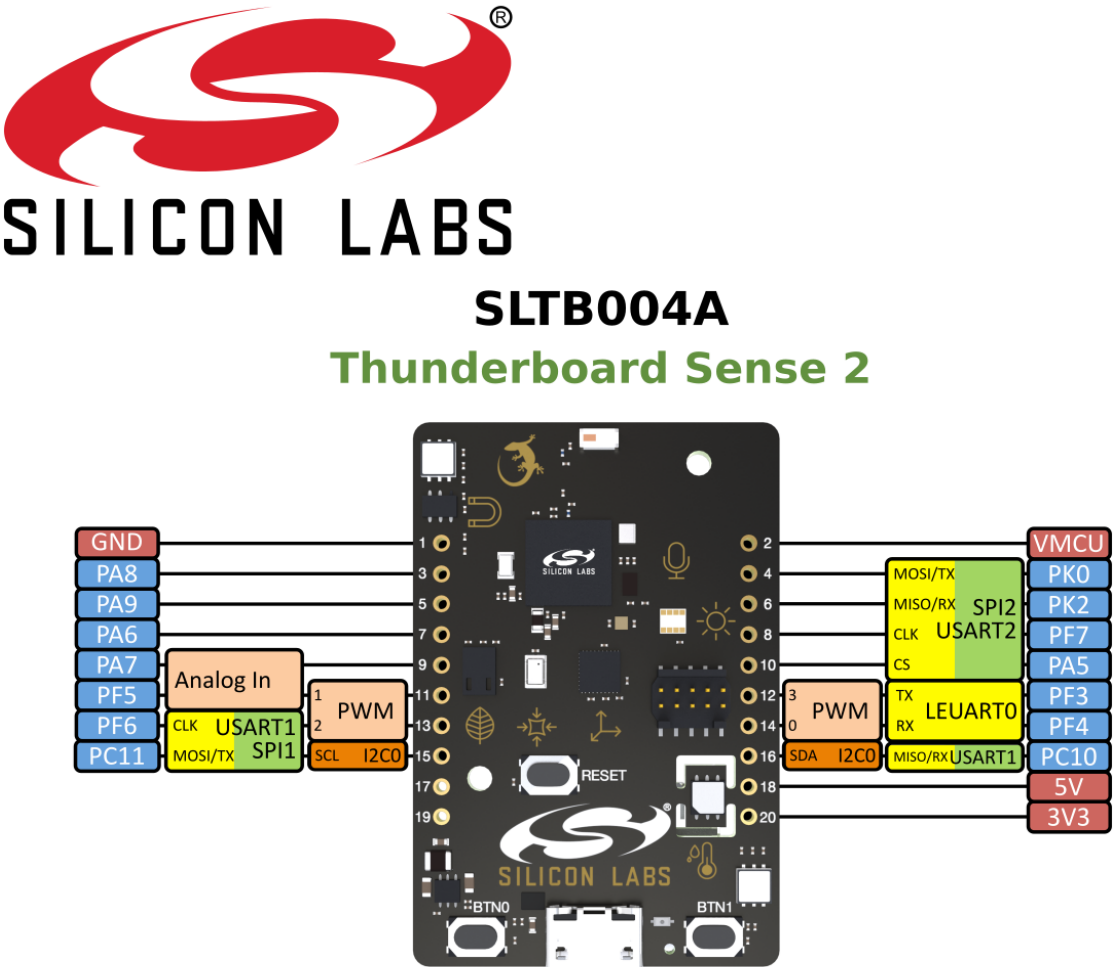
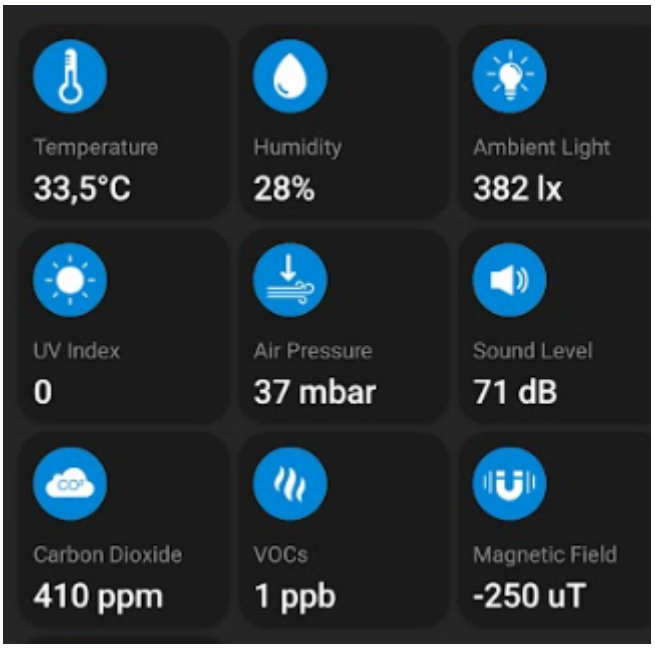
4E-IA3: IoT TP 1



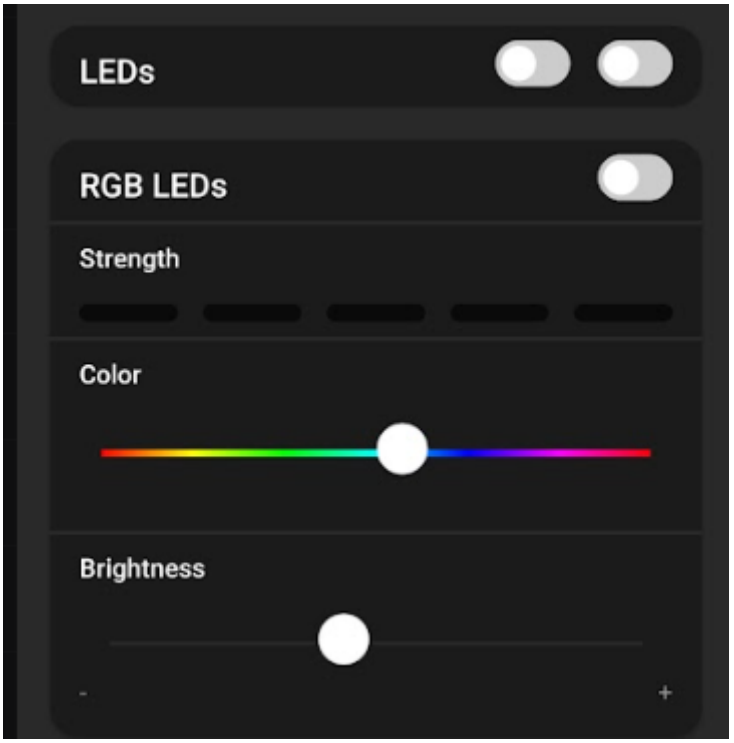
Pinout du Thunderboard Sense 2 de Silicon Labs (source image: os.mbed.com)

NB: le repository du projet est disponible ici: https://github.com/avmolaei/repo_iot

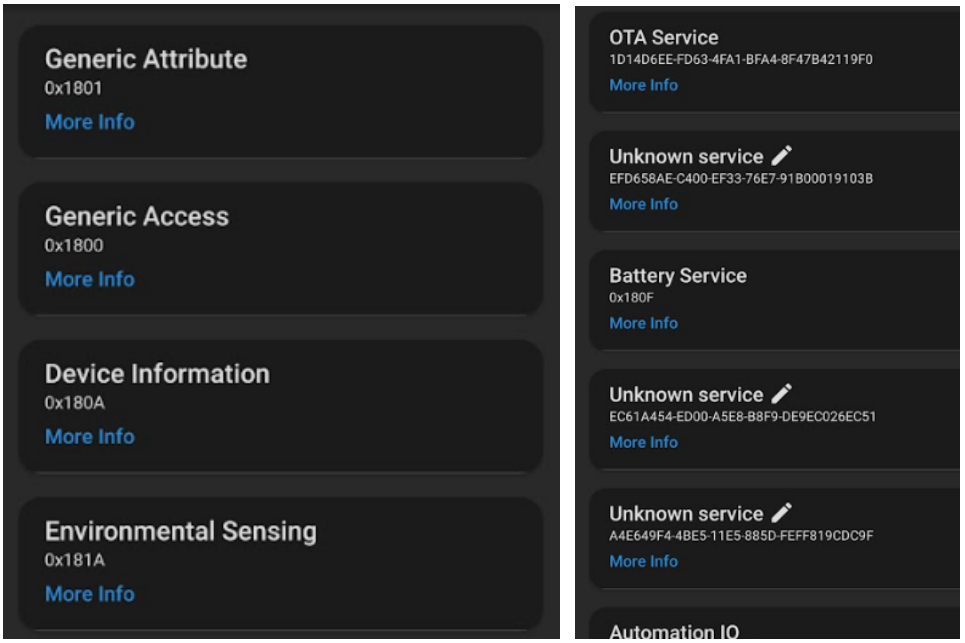
QUESTION 1:
La thunderboard agit comme plusieurs capteurs: luxmètre, capteur de température et d'humidité, accéléromètre, etc:



Cependant, la thunderboard peut être aussi vue comme un actionneur: elle dispose de quelques LED:

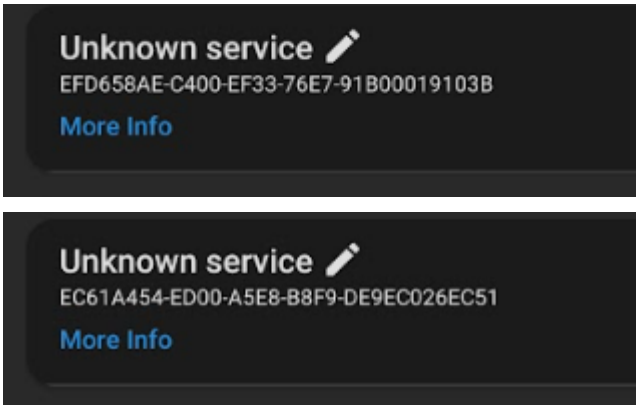


QUESTION 2:
On peut voir plusieurs services:



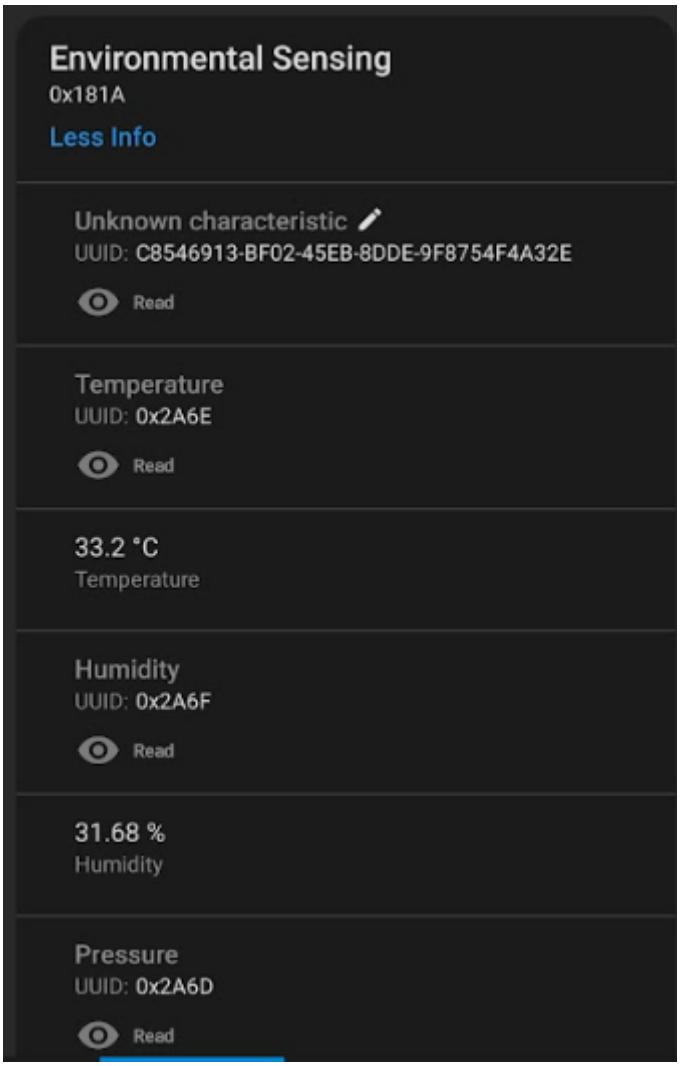
On reconnaîtra les services d'information sur l'appareil (device information), d'accès aux données des capteurs (environmental sensing), et d'accès aux différents IO embarqués à la carte (automation IO)

QUESTION 3:
Les services inconnus ont des UUIDs similaires. Seul l'auteur du code en connaît les fonctionnalités, elles sont masquées.



QUESTION 4:
Les services exposés permettent un accès assez basique aux différents I/O embarqués de la carte, pour une rapide prise en main. C'est logique, il s'agit du but d'un logiciel de démo.

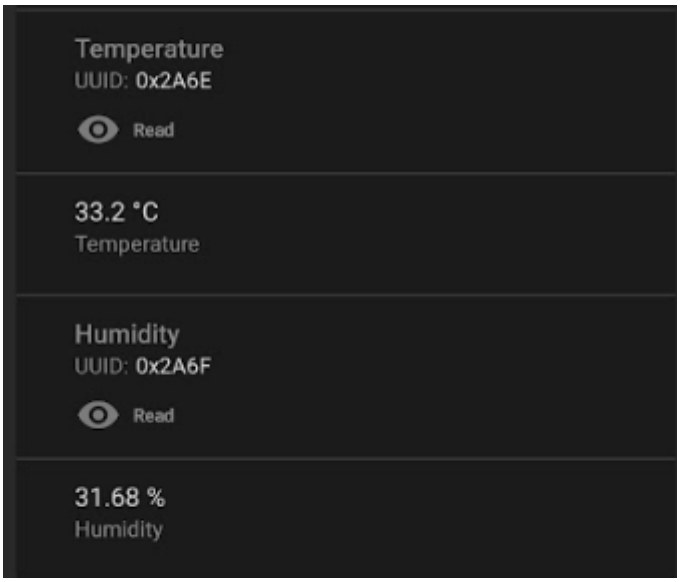
QUESTION 5:
On peut voir, sous environmental sensing, de différentes caractéristiques. Ce sont celles des capteurs de la carte: température, humidité, pression, etc.



QUESTION 6:
Concernant les caractéristiques inconnues (cf capture d'écran ci-dessus), leur format apparaît sous forme d'un UUID beaucoup plus long que la valeur physique envoyée par les différents capteurs (0x2A6E pour la température par exemple).

QUESTION 7:
Le constat est le même que pour les services: certaines caractéristiques sont ouvertes, et connues; d'autres sont propriétaires et fermées, impossibles à décoder sauf si l'on dispose du code source.

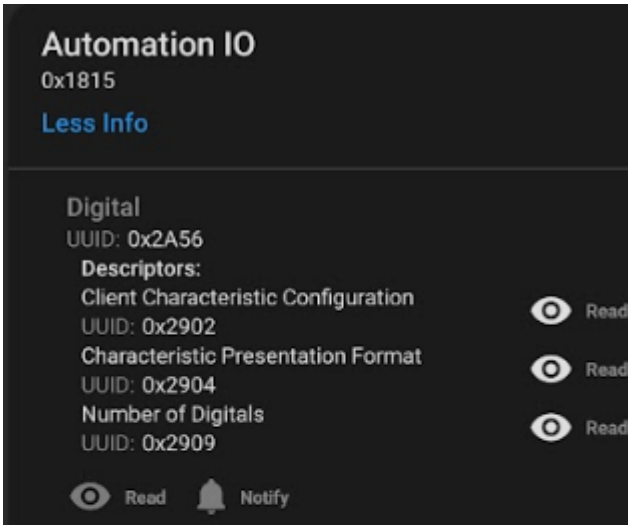
QUESTION 8:
Lorsque l'on clique sur read, on affiche directement la valeur de la température et de l'humidité.



QUESTION 9:
Ici, on vient chercher la valeur du capteur depuis la carte: on est en mode pull.

QUESTION 10:
Ainsi, lorsque l'on clique sur read, c'est le firmware de démo qui vient chercher la valeur du capteur, et qui l'envoie à l'application, qui fait la mise à jour via l'interface graphique.

QUESTION 11:
La première caractéristique "Digital" dispose d'un bouton "Notify" supplémentaire, et d'informations supplémentaires sur ce capteur:



QUESTION 12:
Si l'on appuie sur le bouton, et que l'on clique sur read à nouveau, la valeur de Digital s'actualise, de Inactive à Active. (oubli de capture d'écran : (, désolé)

QUESTION 13:
Du point de vue du MCU, le bouton est un périphérique branché sur un I/O configuré en entrée: il s'agit donc d'un capteur.

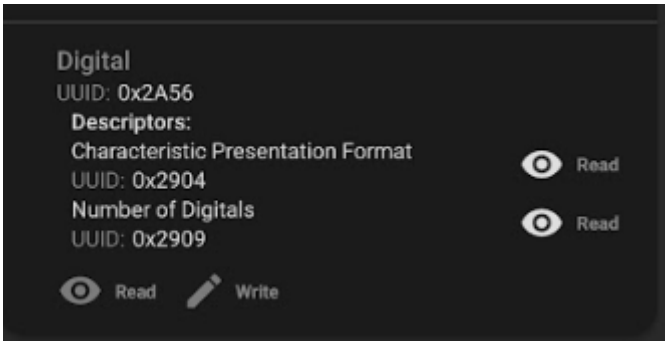
QUESTION 14:
Comme précédemment, les valeurs ici sont pollées.

QUESTION 15:
En appuyant sur notify, on rafraîchit en temps réel l'affichage de la valeur du bouton: plus besoin d'appuyer sur read.

QUESTION 16:
Cette fois-ci, les mises à jour sont pushées: elles arrivent du MCU à l'application.

QUESTION 17:
L'option notify existe grâce au CCCD, le client characteristic configuration, lorsqu'il a pour valeur 0x2902.

QUESTION 18:



Contrairement à la caractéristique précédente, nous avons ici l'option "Write".

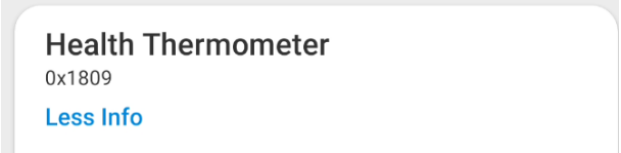
QUESTION 19:
Cliquer sur Write ~~écrit automatiquement le compte rendu des TP d'IoT!~~ allume et éteint la LED.

QUESTION 20:
Du point de vue du MCU, la LED est un périphérique câblée sur un pin I/O configuré en sortie: il s'agit d'un actionneur

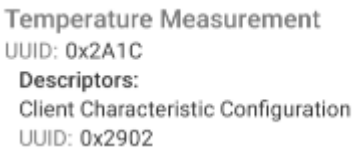
QUESTION 21:
Ici, il n'y a pas besoin de Notify. C'est l'application EFR qui envoie la valeur pour conditionner l'allumage/extinction de la LED. Dans le code de l'application, c'est probablement un buttonListener de la classe java.lang.Object qui déclenche l'envoie de l'appli au MCU des instructions lorsque l'on clique sur le bouton dans la GUI.

(petit changement de smartphone entre temps: on passe d'un Poco M3 Pro à un Pocophone F1)

QUESTION 22:
Le UUID de la fonction thermomètre est dans le service Health Thermometer; il vaut 0x1809.



QUESTION 23:
Le UUID du service Temperature Measurement est 0x2A1C



QUESTION 24:
Lorsque l'on clique sur read, on peut voir dans le champ "value" la manière dont la température est récupérée: "Notifications disabled, Indications enabled" par exemple.

QUESTION 25:
En cliquant sur indicate, on peut lire la température en direct.

QUESTION 26:

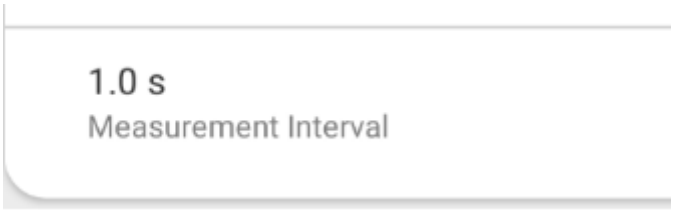
Indicate et Notify semblent fonctionner de la même manière. Indicate fonctionne avec un principe d'ACK, c.a.d un accusé de réception. Le client (l'application) doit valider la bonne réception du paquet. Ceci se fait en software, dans la couche 7 du modèle OSI. Cependant, Notify communique directement le paquet sans passer par vérification. Cette subtilité est importante dans le contexte de l'IoT: les objets connectés se doivent d'être frugaux; on parle d'appareils qui doivent pouvoir tenir plusieurs mois sur une charge d'une petite pile bouton Lithium/alcaline. On pourrait penser à une analogie entre l'UDP et le TCP.

QUESTION 27:
Le sous objet est le CCCD, vu précédemment dans la démo. Son UUID est 0x2902.

QUESTION 28:
La fréquence est actualisée toutes les secondes environ. On dira que la fréquence de rafraîchissement est de l'ordre de 1Hz.

QUESTION 29:
Dans Measurement interval, la période de rafraîchissement est effectivement 1s.

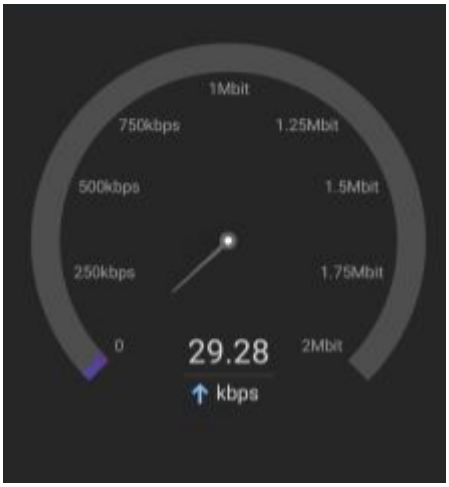
QUESTION 30:
Le lien entre est évident: La fréquence à laquelle les mises à jour se font sont dans la caractéristique measurement interval; qui ici vaut 1s.



(On repasse sur le Poco M3 Pro)

QUESTION 31:
En mode Notification, le débit max atteint environ 300kbps:

QUESTION 32:
En mode Indication, le débit max est environ à 30kbps:



QUESTION 33:
On a mentionné précédemment que l'indicate demande plus de traitement logiciel (couche 7) pour valider une communication (ACK). Ceci explique le débit réduit; car du traitement additionnel est à faire pour accuser la réception de chaque paquet envoyé.