Shu-Qi

Table des matières

[2 Enoncé 3](#_Toc486948684)

[3 Résumé 3](#_Toc486948685)

[4 Remerciement 3](#_Toc486948686)

[5 Projet CHIC 3](#_Toc486948687)

[5.1 Contexte 3](#_Toc486948688)

[5.1.1 Objectifs 3](#_Toc486948689)

[5.2 Coopération 3](#_Toc486948690)

[6 ShuQi 3](#_Toc486948691)

[7 Bluetooth 3](#_Toc486948692)

[7.1 Choix du microcontrôleur 3](#_Toc486948693)

[7.2 Profil Stuff Manager 4](#_Toc486948694)

[7.2.1 Stuff Manager Service 4](#_Toc486948695)

[8 Alimentation 5](#_Toc486948696)

[8.1 USB Type-C 5](#_Toc486948697)

[9 Communications 6](#_Toc486948698)

[10 Application 6](#_Toc486948699)

[10.1 Architecture 7](#_Toc486948700)

[10.2 Etats 8](#_Toc486948701)

[10.2.1 STATE\_SLEEP 8](#_Toc486948702)

[10.2.2 STATE\_READ 8](#_Toc486948703)

[10.2.3 STATE\_RECOGNIZE 8](#_Toc486948704)

[11 Application Smartphone 9](#_Toc486948705)

[11.1 Frameworks 9](#_Toc486948706)

[11.2 Cordova 9](#_Toc486948707)

[12 Tables des sigles et acronymes 11](#_Toc486948708)

[13 Références 11](#_Toc486948709)

[13.1 Bluetooth 11](#_Toc486948710)

[13.2 USB 11](#_Toc486948711)

[13.3 Batteries 11](#_Toc486948712)

# Enoncé

<Enoncé à récup sur: <https://sitehepia.hesge.ch/diplome/ITI/2017/>>

# Résumé

Document écrit à base de « on a fait truc », basé sur 3pers du singulier (il, elle, on)

# Remerciement

Beuchat, Axel Collet, Team CHIC GE, Team CHIC EPFL, etc...

# Projet CHIC

## Contexte

### Objectifs

## Coopération

Ce projet a été fait en coopération avec Axel Collet. Pour de plus amples informations sur les parties suivantes, je vous invite à lire son document:

* RFID
* PCB

# ShuQi

C'est quoi?

# Bluetooth

Ceci est la plus grande partie du projet. C'est elle qui s'occupera de la communication entre la centrale et le smartphone.

Il fallait trouver un moyen de communication entre un smartphone et la centrale. Ce qui nous limite dans les protocoles supportés par la majorité :

* Bluetooth
* Bluetooth Low Energy
* Wifi
* NFC

Le cahier des charges était que l’on puisse communiquer depuis le smartphone vers la base à faible portée. Car il est inutile de vérifier les affaires se trouvant loin de nous.

La portée utile est d’environ trois mètres maximum à

La communication se fera en Bluetooth Low Energy entre le smartphone et la base.

https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.s132.sds%2Fdita%2Fsoftdevices%2Fs130%2Fsd\_resource\_reqs%2Fhw\_block\_interrupt\_vector.html

## Bluetooth Low Energy

La technologie choisie est le Bluetooth Low Energy (Bluetooth 4), ou anciennement appelé le Wibree. Contrairement à la dénomination de version de ce protocole, ce n’est pas le remplaçant du Bluetooth 2.

C’est plutôt que dans la famille du Bluetooth, il y a deux sous catégories complémentaire qui sont :

* Bluetooth Standard
* Bluetooth Low Energy

Ce sont deux technologies complémentaires qui partagent un même nom mais n’ont pas la même utilité. Le Bluetooth Standard est utilisé principalement pour des applications qui demandent un transfert de données soutenu (streaming, musique, etc…) alors que le Bluetooth Low Energy est utilisé pour des transferts de données courts et pas en continu.

De par leur nature différente, le Bluetooth Low Energy est optimisé pour une consommation moindre.

Pour notre projet, on a besoin de demander de temps en temps à la centrale de scanner les objets alentours et de les récupérer sur le smartphone. Au vu de ce constat, le Bluetooth Low Energy colle parfaitement à notre usage !

## GATT

Le BLE fonctionne selon le principe du GATT.

C’est une sorte de serveur qui va recevoir ou envoyer des données.

Les limitations du BLE sont qu’il ne peut pas y avoir de paquets plus grands que 20 octets. De ce fait, il n’est pas possible d’envoyer toute une liste d’un coup. De ce fait il a fallu trouver une solution pour pouvoir lire toute la liste.

## Profil Stuff Manager

Il a fallu créer un profil Bluetooth personnalisé pour ce projet car il n’y en avait aucun prédéfini qui collait à notre cahier des charges.

Pour commencer, on a pris le profil Proximity (PXP) pour lui rajouter un service : Stuff Manager Service.

Le profil Proximity possède trois services :

|  |  |
| --- | --- |
| Link Loss |  |
| Immediate Alert |  |
| Battery |  |

Au début on voulait rajouter un service en plus, mais le résultat final était que quasiment tout le profil était changé. Du coup la création d’un profil entier depuis zéro a été nécessaire.

### Stuff Manager Service

Ce service est la base de la communication entre la centrale et le smartphone. C’est ce service qui va s’occuper de lancer le scan et d’énumérer la liste des tags à proximité.

De ce fait, ce service comporte quatre caractéristiques :

|  |  |
| --- | --- |
| Manager Mode |  |
| Entry Number |  |
| Entry Selection |  |
| Entry Value |  |

C’est sur ce service que l’on peut changer l’état de la centrale avec la caractéristique manager Mode. L’algorithme pour accéder à ce service est le suivant :

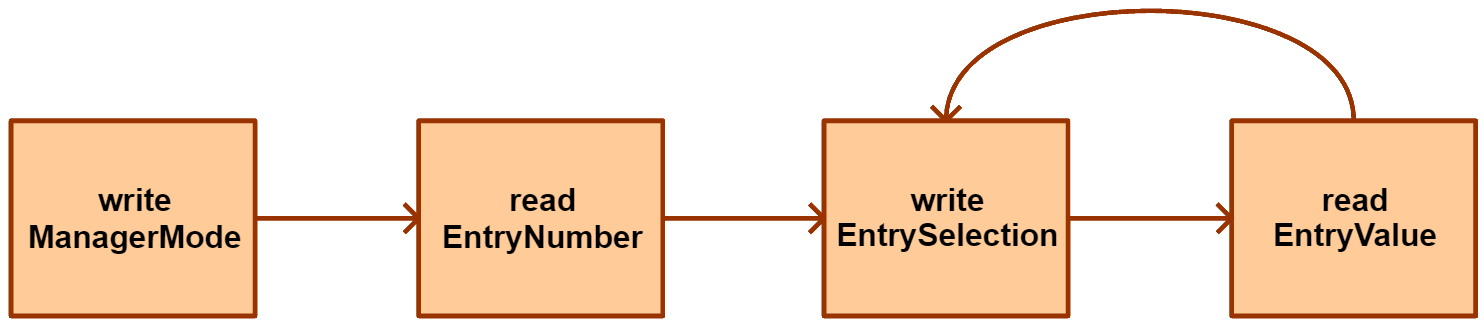


Figure : Algorithme Stuff Manager Service

#### Manager Mode

Cette caractéristique est accessible uniquement en écriture. Et il n’est possible d’écrire que trois valeurs différentes au format *uint\_8* :

|  |  |
| --- | --- |
| MODE\_SLEEP | 0 |
| MODE\_READ | 1 |
| MODE\_RECOGNIZE | 2 |

Ces trois valeurs correspondent aux états de la machine d’états de la centrale. Au vu du schéma de communication retenu entre la centrale et le smartphone, la lecture de cette caractéristique n’était pas utile.

Une amélioration possible c’est de rajouter un accès en lecture de cette caractéristique pour que l’on puisse savoir dans quel état se trouve la centrale. Mais ce n’est pas une fonctionnalité très utile car normalement c’est le smartphone qui va gérer l’état de la centrale.

#### Entry Number

La deuxième caractéristique par ordre d’accès est celle-ci. C’est une valeur au format uint16 accessible en lecture uniquement qui informe du nombre de tags différents qui ont été lu par la centrale.

Sa valeur de base est de 0 et change à la fin du cycle de scanning des tags au nombre de tags qui ont été lus. S’il n’y a aucun tag qui a été lus, la valeur sera de 65535.

#### Entry Selection

Cette caractéristique fonctionne par paire avec Entry Value. Ici elle n’accepte qu’un accès en écriture dans un format uint16.

A chaque écriture, la centrale va commencer par comparer la valeur reçue avec le nombre de tags lus. Si la valeur est plus grande que le nombre de tags lus, elle va changer la valeur de Entry Value par une valeur vide (que des zéros). Sinon elle va aller chercher le bon tag dans sa mémoire et l’écrire sur Entry Value.

#### Entry Value

Ceci est celle on l’on va récupérer l’id du tag RFID demandée sur Entry Selection. C’est une caractéristique accessible uniquement en lecture sur un format de uint128.

Le format utilisé a été choisi sur la base de la taille de l’EPC des tags RFID qui est de 96 bits. Sachant que les formats disponibles sont des multiples de deux, le 128 bits a été choisi.

Le profil contient 2 services : IAS et STM

# Alimentation

La deuxième grosse partie du projet est l'alimentation. Au vu de la nature de ShuQi, ce sera un module qui fonctionnera sur batterie. De ce fait, il y a trois points important :

* La batterie
* Le système de recharge
* L’autonomie

Sur ces trois points, il faut étudier la question de

## USB Type-C

L’USB type-C est un nouveau connecteur qui va succéder aux connecteurs USB-A et micro-USB. C’est un connecteur réversible qui possède 24 pins.

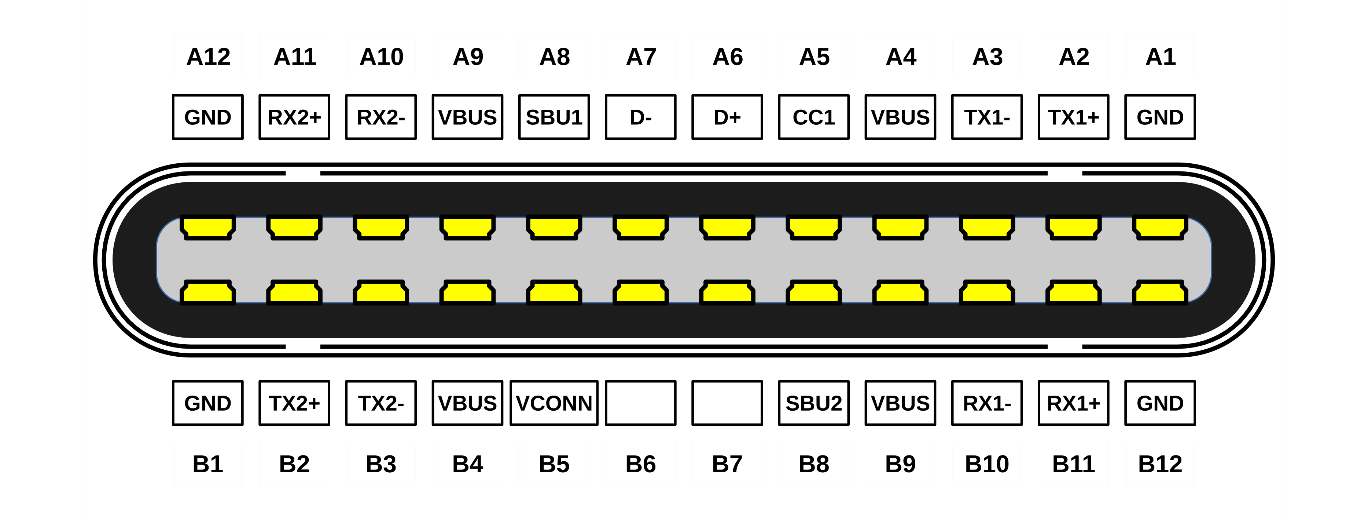


Figure 2: Connecteur USB-C

Je vais utiliser ce connecteur uniquement pour son alimentation. Il servira à recharger la batterie de la base. De ce fait pas besoin de gérer une connexion USB.

L’USB Type-C étant réversible, il a besoin de d’un contrôleur qui permet de savoir le sens de branchement de la prise avec les pins CC1 et CC2 :

< [https://www.digikey.com/-/media/Images/Article Library/TechZone Articles/2017/March/Designing In USB Type-C and Using Power Delivery for Rapid Charging/article-2017march-designing-in-usb-type-c-fig4.jpg?ts=784c69df-ddde-4f4c-a720-5f95599930ce&la=en-SG](https://www.digikey.com/-/media/Images/Article%20Library/TechZone%20Articles/2017/March/Designing%20In%20USB%20Type-C%20and%20Using%20Power%20Delivery%20for%20Rapid%20Charging/article-2017march-designing-in-usb-type-c-fig4.jpg?ts=784c69df-ddde-4f4c-a720-5f95599930ce&la=en-SG)>

Il est possible de faire une recharge rapide, mais on ne va pas l’intégrer ici pour des raisons de couts. Ou pas, quasi aucune différence de prix. On garde la recharge rapide.

# Communications

La solution retenue est que la communication se fera sur la forme d’un schéma maitre-esclave :

<Schéma maitre-esclave>

La base sera en attente de l’arrivée d’une commande de la part du smartphone dans un mode basse consommation. Elle se réveillera à chaque fois que le smartphone va écrire dans une caractéristique.

Au réveil, elle va traiter la nouvelle information récupérée sur la caractéristique changée.

# Application

La centrale est gérée par une logique assez simple. Au vu de son schéma esclave, elle sera la plupart du temps en mode basse consommation. Son réveil ne se fera que de la part du smartphone qui a été appairé.

Pour commencer, la centrale comporte trois états :

* STATE\_SLEEP
* STATE\_READ
* STATE\_RECOGNIZE

L'état de base est STATE\_SLEEP. Comme son nom l'indique, la centrale ne fait aucune action et se mettra en veille. Elle sera uniquement réveillée par des interruptions qui viennent soit du bouton, soit de la part du Bluetooth en cas d'écriture sur une des caractéristiques.

A l'écriture de la caractéristique *Manager Mode*, on va récupérer ce qui a été écrit et on va comparer la valeur pour savoir dans quel mode la centrale doit passer. Cette caractéristique est celle qui a la plus grande priorité, ce qui fait que l'on peut interrompre les opérations pour passer dans le mode que l'on veut.

Mais il y a une limitation, on ne peut pas passer du mode STATE\_READ au STATE\_RECOGNIZE sans passer par STATE\_SLEEP comme marqué sur le schéma suivant.

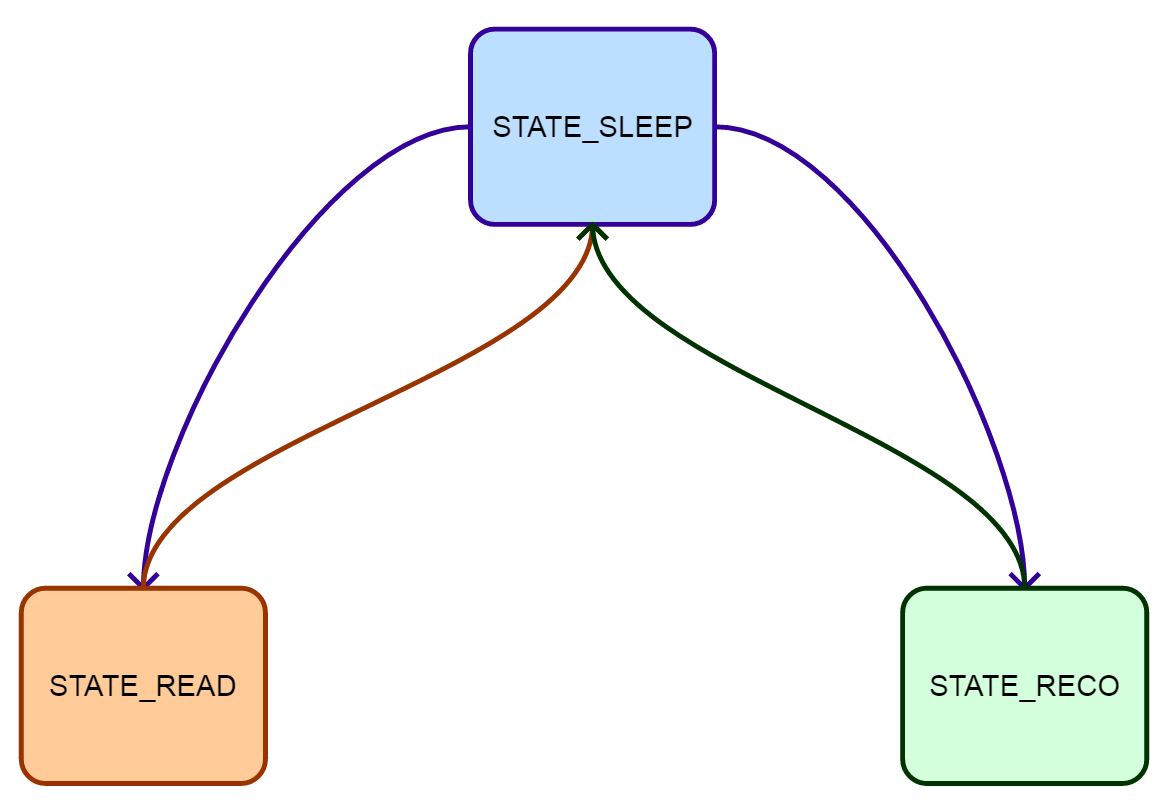


Figure 3: Etats de la centrale

Une explication plus détaillée de chaque état suivra.

Ensuite, vu que l’application sera assez complexe, on va utiliser GCC pour la compilation car Keil est limité à un programme de 32ko avec la licence gratuite. Ce qui n’est pas un grand problème vu que le SDK de Nordic est compatible Keil, IAR et GCC. Par contre, la différence entre Keil et GCC c’est qu’il y a une configuration à effectuer pour avoir un environnement de travail fonctionnel[[1]](#footnote-2).

Comme tout le projet sera généré par un makefile,

## Architecture

L’application va être basée sur le SDK de Nordic qui supporte les microcontrolleurs nRF51 et nRF52. La version utilisée de ce SDK est la stable dernière en date : *V12.3.0*.

La partie des services Bluetooth est générée par un plugin spécifique aux *nRF* sur *Bluetooth Developper Studio*.

## Etats

### STATE\_SLEEP

Cet état est celui ou la centrale va passer le plus de temps, il s'occupe de mettre le processeur en veille.

### STATE\_READ

Celui-ci est l'état le plus important : Il s'occupe de lire les tags à proximité via le RFID, les gardez en mémoire pour les mètres à disposition à travers le service *Stuff Manager*. De ce fait, dès que l'état change, il va allumer le module et lui lancer la commande de scan. Ensuite tous les tags scannés à proximité vont être récupérer dans un buffer et ensuite ils seront mis en RAM jusqu'au prochain changement d'état.

<sch state\_read (genre la suite d'op)>

Bien sûr s’il y a une coupure de connexion entre le smartphone et la centrale pendant cet état, elle va repasser automatiquement dans l’état STATE\_SLEEP.

Par contre il serait possible de rajouter un timeout qui repasse la centrale sur l’état STATE\_SLEEP au cas où l’application du smartphone plante et ne lui redonne pas l’ordre de changer d’état.

### STATE\_RECOGNIZE

Cet état est spécifique et utilisé pour la configuration des tags.

# Application Smartphone

La troisième grosse partie de ce projet est l’application pour les Smartphone. Le marché se décompose principalement entre trois OS : Android, iOS et Windows Phone. La part de marché de chaque OS à 2016Q3[[2]](#footnote-3) est de 86.8% pour Android, 12.5% pour iOS et 0.3% pour Windows Phone. De ce fait, on va se concentrer sur la plateforme Android et il y a un autre avantage, c’est que aucune licence n’est demandée pour pouvoir programmer dessus.

Maintenant, il est possible de créer une application Android de multiples façon différentes :

Android Studio qui est l’IDE proposé par Google qui permet de créer des applications assez simplement si on utilise les objets de base (ce qui implique une interface graphique assez rudimentaire). Par contre le problème c’est que c’est uniquement pour Android.

L’autre solution est de passer par un framework type Cordova qui permet de créer des applications basées sur du HTML5/Javascript et qui permet de les compiler pour plusieurs plateformes.

## Frameworks

Pour le choix du framework, la plupart sont basée sur Cordova

## Cordova

# Tables des sigles et acronymes

|  |  |
| --- | --- |
| BLE | Bluetooth Low Energy |
| RFID | Radio Frequency IDentification |
|  |  |

# Références

## Bluetooth

* Bluetooth SIG

## USB

## Batteries

* <http://batteryuniversity.com/learn/article/charging_lithium_ion_batteries>
* <https://1drv.ms/x/s!AmiKctjbO7jjgoN_3EVrs0gcTp93xA>

1. https://devzone.nordicsemi.com/tutorials/7/development-with-gcc-and-eclipse/ [↑](#footnote-ref-2)
2. http://www.idc.com/promo/smartphone-market-share/os [↑](#footnote-ref-3)