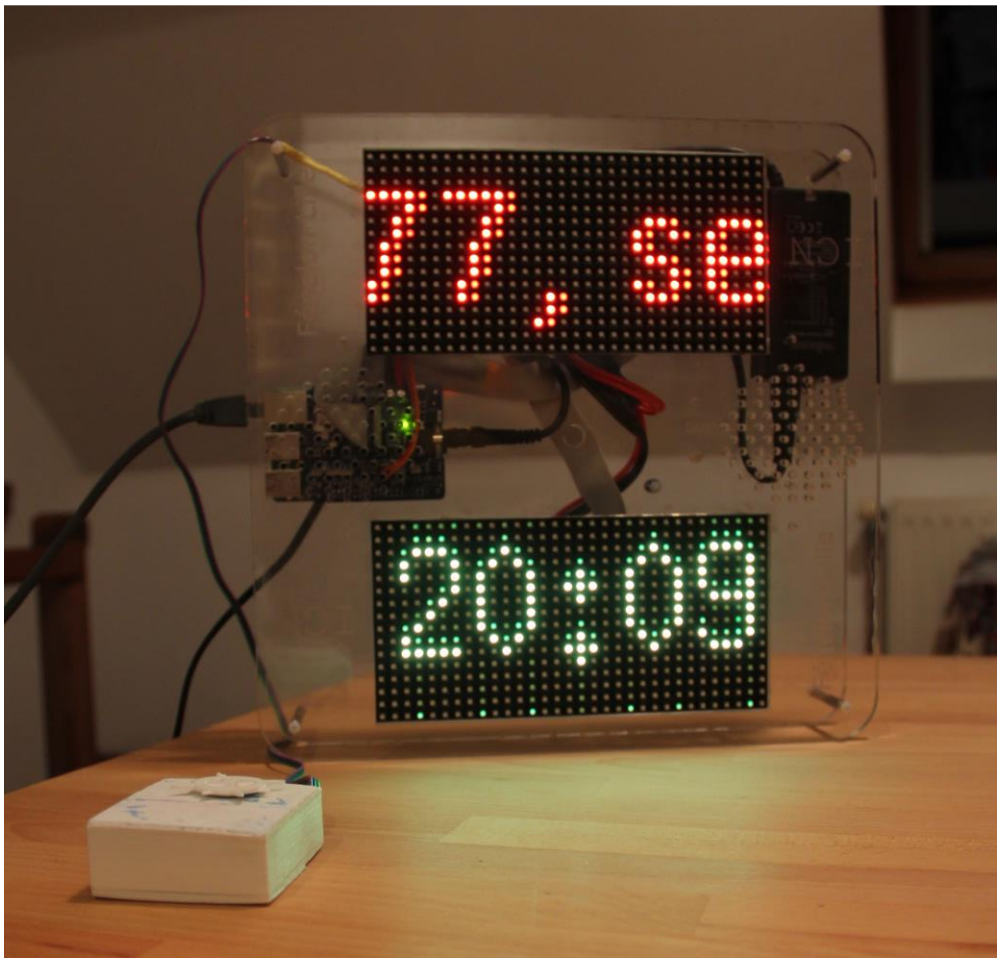
UNIVERSITE DE LILLE

POLYTECH LILLE

Département IMA



Horloge numérique DCF77, serveur de temps et ludique

Rapport de projet du groupe P22

Fabrice TAINGLAND | Amaury KNOCKAERT

Encadrants école : Mr Xavier REDON, Mr Alexandre BOE, Mr Thomas VANTROYS

Année universitaire 2017/2018

SOMMAIRE

INTRODUCTION

Dans le cadre de nos études à Polytech Lille dans la section IMA nous avons été amené à réaliser un projet de 4e année, nous avons choisi un sujet proposé par nos enseignant : L'horloge serveur de temps ludique. Ce projet est la suite d'un projet réalisé par des élèves au lycée Fénelon de Lille, nous avons repris le matériel électronique de ce projet pour développer des fonctionnalités. Notre projet consiste à diffuser l'heure capté via une antenne DCF77 sur les ordinateurs de la salle projet et d'ajouter un côté ludique à l'horloge. Dans une première partie nous vous présentons le projet puis dans les trois parties suivante nous décrivons le travail réalisé pour chaque grande partie du projet.

REMERCIEMENTS

Nous tenons à remercier l'équipe pédagogique de notre formation Informatique, Microélectronique et Automatique (IMA) qui nous a enseigné les savoirs et connaissances utiles et nécessaires pour la réalisation de ce projet. Ensuite, nous remercions également nos tuteurs de projet, Monsieur Alexandre BOE, Monsieur Xavier REDON et Monsieur Thomas VANTROYS pour leur aide et conseils tout au long du projet. De plus, nous souhaitions remercier les responsables du Fabricarium pour leur aide et leurs conseils sur la conception mécanique de notre projet. Enfin, un merci particulier à Monsieur Laurent ENGELS pour le temps qu'il nous accordé pour la réalisation de notre vidéo, ainsi qu'à Monsieur Thierry FLAMEN pour la réalisation du PCB ainsi que les conseils qu'il a pu nous prodiguer.

I - Présentation de l'horloge, des objectifs et des choix techniques

Dans cette première partie du rapport nous allons introduire l'horloge. A travers les objectifs fixés par le cahier des charges que vous allez pouvoir retrouver par la suite, nous allons vous présentez les caractéristiques qui font que cette horloge est dite « connectée ». Enfin, nous allons vous expliquer les démarches qui ont été entreprises pour effectuer les bon choix techniques.

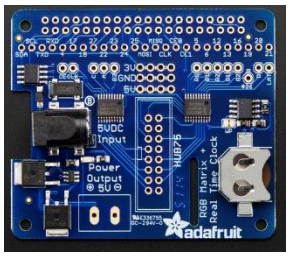
1. L'horloge connectée

L'horloge connectée sur laquelle nous avons travaillé est issu d'un projet réalisé par des élèves du lycée Fénéllon de Lille (59). Ce projet était de base constitué ainsi :

- Une Raspberry Pi 3
- Une alimentation 230V-5.2V AC/DC
- Deux matrices de LED « Adafruit 16x32 RGB LED matrix panel »
- Un shield « Adafruit RGB matrix HAT + RTC »
- Un module 641138, récepteur DCF77

Raspberry Pi 3B

Module DCF77



Adafruit RGB matrix HAT + RTC

Le but de ce projet était de réaliser une horloge dite connectée. Les horloges connectées font partie du nouveau monde dit des objets connectés. Ces objets renfermant un système électronique communiquent en permanence en réseau avec d'autres entêtées. La communication s'effectue soit par internet, soit par Bluetooth, ou encore par onde radios. Cette capacité de faire

communiquer des éléments, pour la plupart de petite taille, a permis à une vaste faune d'objets de la vie quotidienne d'accéder aux réseaux mondiaux. C'est le cas des horloges connectées. Elles ont la faculté d'accéder à diverse informations, mais surtout d'en diffuser. Dans notre cas l'horloge reçoit l'heure par ondes radios depuis une horloge atomique située à Mainflingen près de Francfort en Allemagne, dans le but de l'afficher sur une matrice de LEDs. Avec tous les éléments à notre disposition, nous étions alors en capacité d'ajouter des fonctionnalités à cette horloge pour la rendre plus interactive, que ce soit avec des réseaux ou avec l'utilisateurs.

2. Objectif

Les objectifs du projet on était défini en 6 points distinct que voici :

- Installer un serveur de temps NTP (**N**etwork **T**ime **P**rotocol).
- Améliorer la réception DCF77.
- Protéger le système contre les coupures de courant ou les débranchements inopinés.
- Trouver une application à la 2e matrice de LED.
- Gérer les fuseaux horaires
-

1.1.1. Serveur de temps

L'un des objectifs principaux de notre projet consiste à installer un serveur de temps pour les ordinateurs de la salle de projet, afin que l'heure ne soit pas issue des serveurs de Polytech mais d'une sources plus fiable. L'antenne de DCF77 présent en Allemagne envoie l'heure issue d'une horloge atomique.

Les ordinateurs de la salle projets se connecterons au serveur NTP de l'horloge, pour accéder à l'heure atomique.

1.1.2. Améliorer de la réception DCF77

La réception DCF77 qui s'effectue sur la fréquence de 77.5 kHz n'est pas aussi simple qu'une réception d'un signal wifi ou Bluetooth, notre objectif est de définir par quelle manière réceptionner au mieux. Il nous faut par la même occasion créer un programme de décodage de l'heure.

1.1.3. Protéger l'horloge des coupures brusque

Comme tout appareil électrique il est possible qu'il soit débranché, par inadvertance ou à cause d'une coupure de courant. Ce qui n'est pas embêtant dans la cas d'une horloge classique mais problématique dans note cas car l'horloge comporte un système d'exploitation temps réel qui tourne en boucle. Pour ne pas corrompre le matériel mais aussi les fichiers nécessaires au bon fonctionnement, il faut une extinction du système avant de couper l'alimentation.
Il faut donc mettre en place un système équivalent à un « onduleur » d'ordinateur avec une batterie de secours.

1.2.1. Application deuxième panneau de LED

Sur l'horloge il y a deux matrice de LED une qui affiche l'heure et une autre qui doit participer à l'interaction entre l'utilisateur et l'horloge.

1.2.3. Gestion des fuseaux horaire

L'antenne DCF77 envoie l'heure avec le fuseau horaire de l'Allemagne. Dans le monde il y a 23 autres fuseaux horaire, si l'horloge est installée dans un fuseau horaire différents de celle de l'Allemagne il faut pourvoir le gérer.

3. Choix technique

1.3.1. Protéger l'horloge des coupure brusque

Nous avons choisi d'utiliser une batterie de secours, pour éviter que les débranchements brutaux ne détériorent Raspbian, l'OS présent sur la Raspberry.

Nous avions plusieurs problématiques à gérer :

- Savoir quand l'horloge est débranchée
- Switcher de manière instantané de l'alimentation secteur, à la batterie
- Gérer le boot de Raspbian lors du branchement de l'horloge
- Gérer la recharge de la batterie.
- Gérer la puissance des LEDS

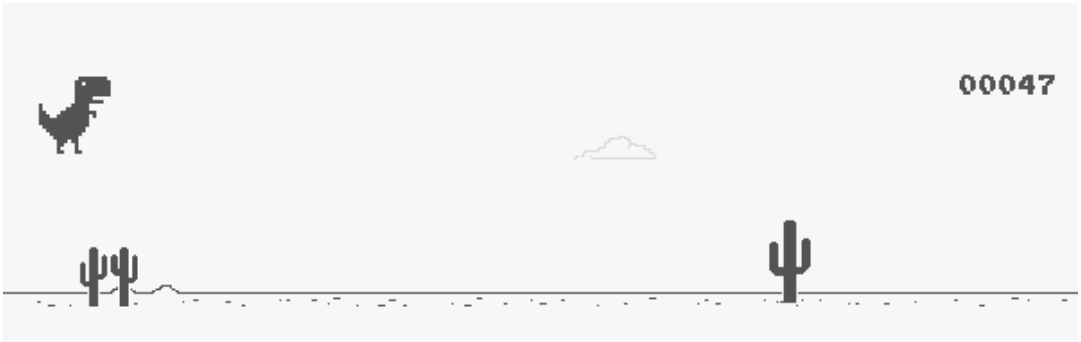
Pour savoir si l'horloge est alimenté avec la prise secteur, nous avons choisi de mettre une pin du GPIO sur l'entrée secteur de la Raspberry. Si la tension est à 5 v l'alimentation est branché sinon on commande l'extinction. Pour switcher d'une alimentation à une autre nous avons choisi d'utiliser une diode. Lorsque le niveau de tension de l'alimentation secteur est plus petit que celui de la batterie alors la diode passe de bloquante à passante et la batterie alimente la Raspberry .

Pour gérer le rallumage de Raspbian, nous avons opté pour un système qui s'éteint entièrement, on coupe l'alimentation de la Raspberry lorsqu'elle s'est éteinte proprement. Pour la redémarrer il suffit de la rebrancher. Pour charger la batterie, nous avons choisi d'utiliser un gestionnaire de charge intégré.

Du côté des deux panneaux de LEDs, ils consomment 20 watts, si on souhaite les alimenter durant l'extinction de la Raspberry il nous faudrait une batterie avec puissance instantané d'environ 25w donc plus grosse et plus chère que si on alimente uniquement la Raspberry. Nous avons choisi de ne pas alimenter les Leds lorsque l'horloge est débranchée. A noter que ce choix nous a implicitement mené vers le choix d'éteindre la Raspberry en cas de coupure, et non de la laisser allumée.

1.3.2. Application deuxième panneau de LED

Pour animer le deuxième panneau de LED, nous avons fait le choix de développer un jeu commandé par téléphone. Nous nous sommes inspiré du jeu présent sur le navigateur Google Chrome, disponible lors d'une erreur de connexion en absence de connexion internet.
Le jeu consiste à sauter au-dessus de cactus avec un dinosaure le plus longtemps possible :



La commande du saut se fait via un bouton Jump présent sur l'application.

Comme seconde interaction nous avons mis en place un bandeau d'informations sur la même matrice de LED que le jeu. Via l'application nous envoyons le lien d'un flux RSS ou ATOM, la Raspberry se charge de télécharger le contenu et de l'afficher sur la matrice de LED.

1.3.3. Gestion des fuseaux

Pour gérer les fuseaux horaires, nous avons choisi de le faire via l'application, l'utilisateur renseigne uniquement le décalage horaire entre sa zone et le méridien de Greenwich.

II - Application : jeu, flux rss, fuseaux horaire

Une des fonctionnalités de l'horloge est son aspect ludique. C'est pour cela que nous avons développé un jeu commandé depuis smartphone, un affichage de flux RSS ou de message, ainsi qu'un paramétrage des fuseaux horaires. La communication est simple et repose sur l'envoi d'une trame ayant pour donnée : un code sur un octet, suivi de l'information.

Notre choix s'est porté sur une application de smartphone. Nous avions, un temps exploré la possibilité de réaliser une application web mais celle-ci rendait compliqué l'utilisation du Bluetooth pour communiquer entre les deux bouts.

Pour échanger des données entre l'horloge et l'application nous avons choisi la technologie Bluetooth. Nous avions aussi à disposition, la technologie Wifi et la transmission par onde radio. Notre choix s'est porté sur la technologie Bluetooth car elle répondait aux critères suivant :

- Proximité : l'utilisateur interagis toujours avec l'horloge en étant à proximité. Il ne joue pas au jeu en étant à l'autre bout de la planète.
- Simplicité : Le Bluetooth est facilement paramétrable sur Android. La Raspberry intègre d'office une puce Bluetooth. Le protocole RFCOMM du Bluetooth permet d'envoyer facilement de petites trames de données.

1. Connexion

Pour mettre en place la connexion Bluetooth sur l'application, nous avons fait appel aux différentes bibliothèques Android :

```
android.bluetooth.BluetoothAdapter;
android.bluetooth.BluetoothDevice;
android.bluetooth.BluetoothSocket;7
```

Elles permettent respectivement de gérer le module Bluetooth du téléphone, un appareil Bluetooth associé et les sockets Bluetooth

2.1.1. Bluetooth Adapter

Lorsqu'on a besoin d'une connexion Bluetooth dans l'application (vue du jeu, flux rss et fuseaux horaires) on crée un thread de connexion. Mais avant cela on va vérifier que le Bluetooth est activé. Pour cela nous avons utilisé le code suivant

```
bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (!bluetoothAdapter.isEnabled()) {
    startActivity(new Intent( packageContext: this, Bluetooth.class));
}
```

Si le Bluetooth n'est pas activé on renvoie l'utilisateur vers la page Bluetooth qui explique la démarche à suivre et force l'utilisateur à activer le Bluetooth via la méthode :

```
startActivityForResult(enableBlueTooth, REQUEST_CODE_ENABLE_BLUETOOTH);
```

2.1.2. Thread de connexion

Une fois le Bluetooth activé et que l'horloge est bien appairée au téléphone via une recherche dans la liste des appareils, renvoyés par la méthode :

```
bluetoothAdapter.getBondedDevices();
```

On cré un thread de connexion qui va se charger, en tâche de fond, de maintenir la connexion. C'est un enfant de la Class thread qui comporte la méthode Run dans laquelle la connexion se fait :

```
public void run() {
    bluetoothAdapter.cancelDiscovery();

    try {

        mmSocket.connect();
    } catch (IOException connectException) {


        try {
            mmSocket.close();
        } catch (IOException closeException) {
            Log.e(TAG,  msg: "Impossible de fermer le client socket", closeException);
        }
        return;
    }
```

Une méthode cancel permet de fermer la socket. C'est aussi dans cette class qu'on va retrouver les méthodes qui permettant d'écrire et de lire sur la socket, on utilise les méthodes issues de la class BluetoothSocket :

```
public void write(byte[] data) {
    try {
        Log.i(TAG, msg: "WRITE NEW MESSAGE: "+data.length);
        mmSocket.getOutputStream().write(data);

    } catch (IOException e) {
        Log.e(TAG, msg: "Exception during write", e);
    }
}
public void read(byte[] data) {
    try {
        Log.i(TAG, msg: "READ NEW MESSAGE: "+data.length);
        mmSocket.getInputStream().read(data);

    } catch (IOException e) {
        Log.e(TAG, msg: "Exception during read", e);
    }
}
```

2.1.3. Connection Bluetooth côté horloge

Du côté horloge, le programme se situe dans */root/horloge* et se nomme *Horloge.cpp*. La bibliothèque Bluetooth se situe dans */root/horloge/Bluetooth/server.* Ce programme lance dans un nouveau thread, une fonction « communication() ». Celle-ci lance une socket d'écoute, lorsque la socket d'écoute trouve une connexion, le programme passe sur la socket de dialogue et coupe la socket d'écoute. Nous faisons cela pour éviter que plusieurs utilisateurs ne se connectent à l'horloge en même temps. Nos tests ont montré que même si la file d'attente est de taille 1, tout se passe comme si les clients en attente voient leurs messages stocké dans un buffer, puis libéré tous en même temps quand le premier client quitte la communication, dans la fonction « recv() », chargé de recevoir les messages entrants.

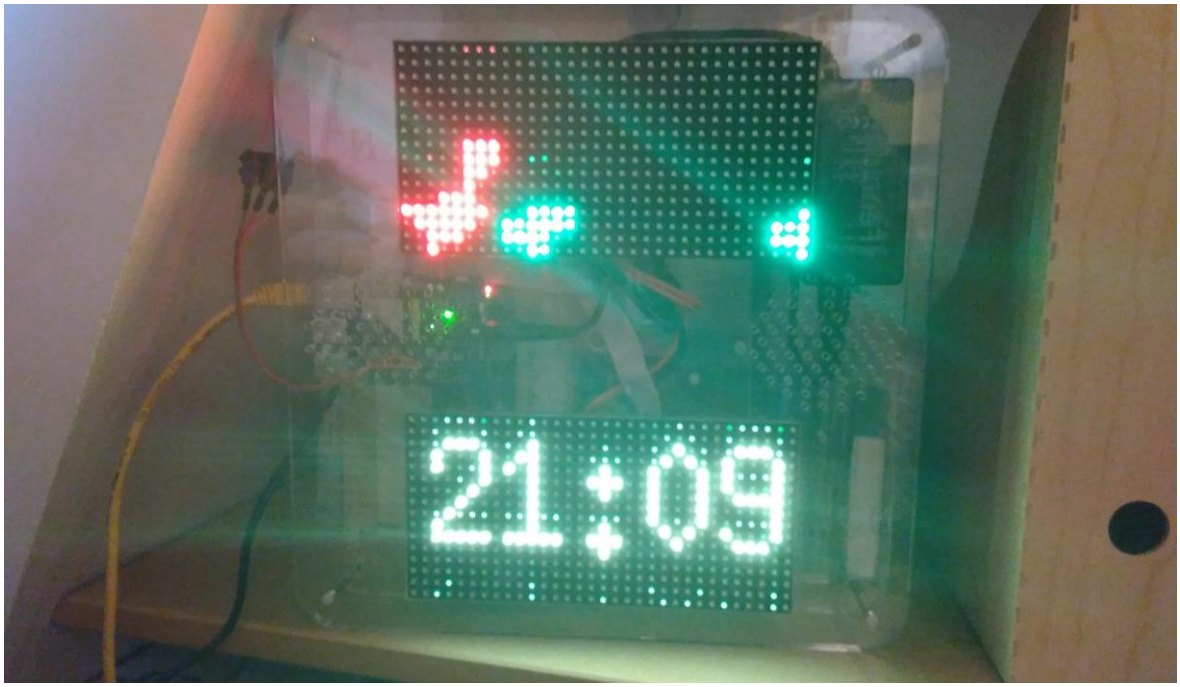
Pour gérer l'envoi et l'écoute de messages Bluetooth en simultanée, nous créons un thread d'envoi qui s'exécute en parallèle du programme de réception. Les messages reçus et les messages envoyés sont stockés dans 2 structures distincts :

```
struct bt_msg_recv {
        uint8_t client;         //est à 1 si un client est présent, sinon 0
        char *msg;    //contenu du message bluetooth reçu ou à envoyer
};

struct bt_msg_send {
        uint8_t client;         //est à 1 si un client est présent, sinon 0
        uint8_t write;          //si il vaut 1, alors la donnée est à écrire, si il vaut 0 elle a été écrite
        char *msg;    //contenu du message bluetooth reçu ou à envoyer
};
```

Pour accéder en dehors de la fonction « communication() », aux messages Bluetooth reçu et envoyés il suffit de lire ou d'écrire dans le champ « msg ». Ces structures sont liées à des mutex permettant ainsi d'être accessibles depuis n'importe quel thread, tout en évitant les conflits d'accès.

2. Jeu



Le jeu présent sur l'écran de l'horloge est une réplique du fameux jeu de google chrome qui consiste à sauter au-dessus de cactus le plus longtemps possible.

2.2.1. Partie Application

Il y a 3 grande partie : la commande du jeu, la gestion de la fin du jeu, l'affichage du score. La commande du jeu consiste à envoyer un caractère via la socket Bluetooth à l'horloge, ce caractère peut signifier le début du jeu ou bien la commande d'un saut :


```
public void btn_jump(View view) {

    if (!jeu_actif) {
        mConnectThread.write(start_jeu);
        txt_information.setText("Sauter au dessus des Cactus");
        jeu_actif = true;
        cmp=0;
       read_socket calcul=new read_socket();
        calcul.execute();
    }
    else {

        cmp++;
        mConnectThread.write(Jump);
    }
```

On peut voir ici la vue de la commande du jeu avec le bouton de saut ainsi que les méthodes java permettant de gérer l'écriture et la lecture de la socket. Pour la lecture nous avons fait une tache asynchrone qui s'éxecute à chaque début de jeu et va lire la socket. Quand on reçoit la requete de fin de jeu, on arrete le jeu et on affiche le score.

2.2.2. Partie Horloge

Dans la partie horloge nous avons un programme qui gère les différentes requêtes venant de l'application, pour le jeu, pour les messages et pour le réglage des fuseaux horaires. Lorsque on reçoit une requête de jeu, celui-ci démarre sur le panneau de LEDs supérieur. Ensuite les cactus arrivent de la gauche et défilent de manière aléatoire vers le dinosaure. La touche de saut envoi un code à l'horloge qui se charge d'animer le dinosaure pour le faire sauter au-dessus des cactus. La partie se termine lorsque le dinosaure entre en collision avec un cactus.

Pour gérer l'affichage des dinosaures et des cactus nous avons un tableau 3 dimensions Largeur * hauteur *couleur Led, l'affichage se fait à chaque actualisation de la matrice à une fréquence définit par un délais d'attente. Après chaque attente toutes les cases contenant du vert (les cactus) se déplacent vers la gauche.

À chaque appuie sur le Bouton Jump de l'appli une requête arrive, on déplace le dinosaure vers le haut on attend trois déplacement cactus puis le dinosaure descend.

Pour vérifier qu'il n'y est pas de collision après chaque déplacement on vérifie qu'il n'y est pas de pixel rouge et vert au même endroit.si c'est le cas on arrête le jeu.

3. Flux RSS/Message

L'aspect ludique de l'horloge n'est pas limité au jeu du dinosaure, il y a aussi la possibilité de mettre des messages ou le contenu d'un flux RSS. Le contenu du message du flux est diffusé au même endroit que le jeu c'est-à-dire sur la matrice de LED supérieur

2.3.1. Partie Application

Sur l'application, nous avons une vue comprenant une zone d'édition de texte et un bouton pour envoyer la donnée de la zone de texte, à l'horloge. Cette zone de texte peut-être un message personnel ou un lien hypertexte vers un flux RSS ou ATOM.


```
public void btn_send(View view) {
    String tmp="";
    tmp=edit_flux.getText().toString();
    if(tmp.isEmpty())
    {
        txt_information.setText("La boite de  texte  est vide ");
    }
    else
    {
        mConnectThread.write(fin_flux);
        byte tempo[] =tmp.getBytes();
        byte envoie[]=new byte[tempo.length+1];
        for(int i =0; i<tempo.length;i++)
        {
            if ( i==0)
            {
                envoie[0]=code_flux;
            }
            envoie[i+1]=tempo[i];
        }
        mConnectThread.write(envoie);
        edit_flux.setText("");
        txt_information.setText("Copier votre  lien dans la fenetre de texte si dessous \n puis cliquez sur envoyer");
    }
```

2.3.2. Partie Horloge

Du côté de l'horloge c'est la fonction « bandeau() », présente dans le fichier ledPanel.c qui gère l'affichage des messages. Ceux-ci sont réceptionnés par le thread d'écoute et envoyé en paramètres à la fonction. Si le message commence par « http:// » ou « https:// » l'horloge interprète ce message comme un lien hypertexte et va alors récupérer le contenu de la page située à l'adresse indiquée. Si le contenu récupéré correspond à un flux RSS ou ATOM, alors les titres en sont extraits et se mettent à défiler sur le panneau de LEDS du haut. Les flux RSS/ATOM sont des fichiers structurés XML permettant de transmettre des informations, généralement des billets de blogs. On peut les trouver sur différents sites, comme ceux traitant de l'actualité par exemple.

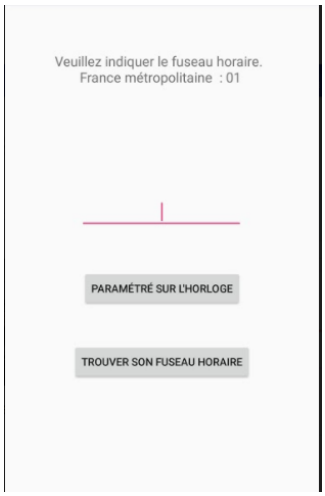
La récupération des titres n'est possible que si l'horloge est connectée à un réseau, lui-même connecté à internet, sans proxy. Les tests réalisés chez nous ont été réussi, cependant nous ne sommes pas parvenues à récupérer le contenu d'un flux à travers le proxy de Polytech Lille.

4. Gestion des fuseaux horaires

Afin de modifier le fuseau horaire de l'horloge. Nous avons créé une vue qui permet à l'utilisateur d'indiquer son fuseau horaire. Il est envoyé ensuite sur l'horloge pour adapter l'heure reçue par l'antenne DCF77.

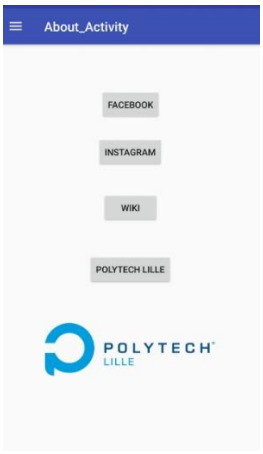
2.4.1. Partie application

La partie application est semblable à celle du flux rss/message, c'est une zone de texte où l'on indique le fuseau horaire choisit.. On a aussi un bouton amenant vers une page internet où l'on peut trouver les fuseaux horaires de son pays.



2.4.2. Partie Horloge

Afin de modifier les fuseaux horaires, il a fallu modifier quelques fichiers de configuration Linux. La gestion des fuseaux horaires sur Linux s'effectue soit via la variable globale 'TZ' soit via le contenu du fichier */etc/localtime.* Ces deux entités stock le fuseau horaire. Pour l'OS Raspbian de la Raspberry Pi c'est le second cas. Pour changer le fuseau horaire il suffit simplement de supprimer ce fichier (en ayant pris soin d'en faire une copie au cas où) et de le remplacer par un lien symbolique pointant vers le contenu du répertoire */usr/share/zoneinfo/Etc/*. Ce répertoire contient une multitude de fichiers nommé *GMT+1, GMT-1, GMT+2,…* Le lien symbolique pointera vers le fuseau horaire souhaité. Dans le cas de la France et de beaucoup de pays Européen c'est le fuseau GMT+1.



Menu "A propos"

Informations sur l'horloge

5. Navigation, à propos et support

L'application comporte également des parties moins techniques et qui ne sont pas en lien avec l'horloge, comme la barre de navigation.

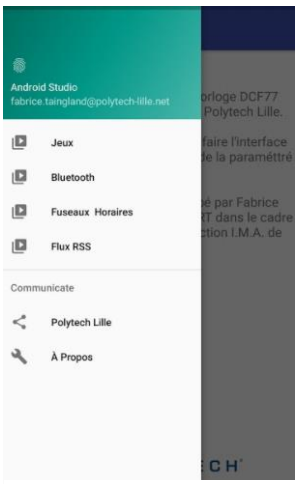
Elle permet depuis n'importe qu'elle vue de se déplacer dans n'importe qu'elle autre vue.Ce système nécessite de gérer le faite qu'on n'ouvre pas deux fois la même vue , par exemple deux vue jeux ouverte.

Pour cela il faut indiquer sur l'activity dans le fichier manifest.xml :

```
android:launchMode="singleInstance"
```

Nous avons aussi développé deux autre vue , une permettant d'accédé au différents reseaux sociaux de polytech , au site du wiki et de polytech, cela permet à l'utilisateur de s'informer sur polytech et de connaître un peu mieux notre projet.

La dernière vue est une page d'accueil introduisant le principe de l'horloge.



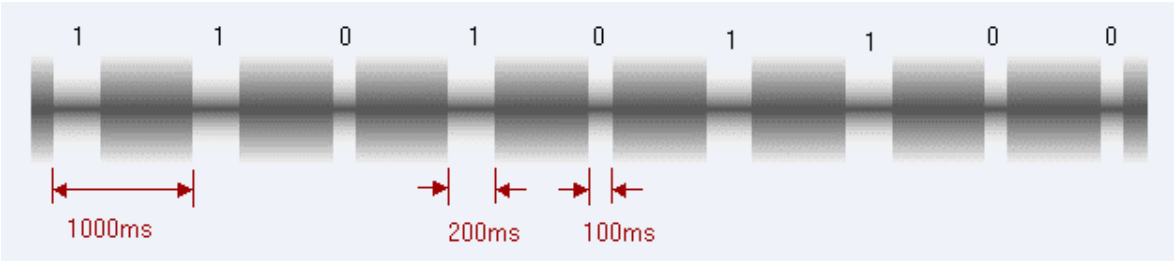
La navigation au sein de l'application

III - DCF77, décodage des données

1. Fonctionnement

Une des parties les plus importante de l'horloge est la réception de l'heure. Nous recevons l'heure grâce à un signal radio envoyé depuis la ville de Mainflingen en Allemagne.

Le signal DCF77 est modulée en amplitude. La durée de transmission d'une trame est de 60 secondes. Un état logique haut correspond à un maintien constant de l'amplitude pendant une durée de 200 ms. Pour un état logique bas, l'amplitude est constante, à l'état haut sur environ 100ms.



Lorsque la trame débute l'émission elle envoie la date atteinte lors de la transmission du dernier bit. La trame se compose de 60 bits. Chaque bit transporte une information sur la date, mais aussi pour l'Etat Allemand en cas de force majeure, nous ne prenons évidemment pas en compte ces bits. La table de décodage de la date est la suivante :

Bit0 :	Début de trame (bit à 1).
Bit1 à 14 :	Réservé pour une utilisation future.
Bit 15 :	Bit d'antenne (0 = antenne normal 1 = antenne de réserve)
Bit 16 :	Changement heure d'hiver/heure d'été (1= passage heure été/hiver)
Bit 17 :	0: inactif / 1: heure d'été (GMT+2)
Bit 18 :	0: inactif / 1: heure d'hiver (GMT+1)
Bit 19 :	1 quand une minute de 61 secondes est insérée
Bit 20 :	Bit de début (start) (Toujours à 1)

Bit 21..27 : Valeur des Minutes (Bit: 21 = 1, 22 = 2, 23 = 4 , 24 = 8 , 25 = 10 , 26 = 20, 27 = 40) Bit 28 : Parité pour tous les bit transmis minutes (Bit de parité paire) Bit 29..34 Valeur des heures (Bit: 29 = 1, 30 = 2, 31 = 4 , 32 = 8 , 33 = 10 , 34 = 20) Bit 35 : Parité pour tous les bit transmis heures (Bit de parité paire) Bit 36..41 : Valeur du jour (Bit: 36 = 1, 37 = 2, 38 = 4 , 39 = 8 , 40 = 10 , 41 = 20) Bit 42..-44 : Valeur du jour dans la semaine (Bit: 42 = 1, 43 = 2, 44 = 4) Bit 45..49 : Valeur du mois (Bit: 45 = 1, 46 = 2, 47 = 4 , 48 = 8 , 49 = 10) Bit 50..57 : Valeur de l'année (Bit: 50 = 1, 51 = 2, 52 = 4 , 53 = 8 , 54 = 10 , 55 = 20, 56 = 40, 57 = 80) Bit 58 : Parité pour tous les bits transmis (Bit de parité paire)

Le programme de d'écoute et de décodage DCF77 se situe dans le dossier */root/DCF77/*. Nous y trouvons un ensemble de fonctions qui permettent de lire les pins GPIOS, ainsi que le programme de mesure et de décodage.

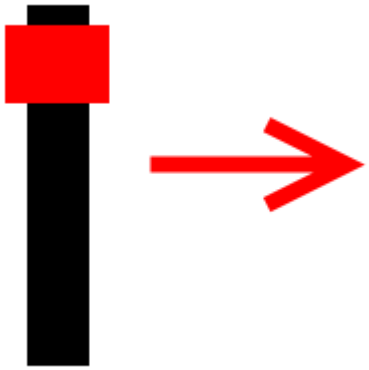
Pour récupérer l'heure nous relions la sortie non-inverseurs du module 641138 à une pin inutilisée par le shield. Nous écoutons cette pin pendant une durée de 120 secondes pour être sûr d'avoir récupéré dans un tableau de taille 1200, une trame complète. Le tableau est de cette taille car nous effectuons une mesure sur une période 100 ms. De cette manière nous récupérons 2 bits à 1, correspondant à l'état haut. Ou alors 1 bit à 1, correspondant à l'état haut bas.

Par la suite une fonction va extraire une trame complète du tableau, tester les bits de parités. Dans le cas où ceux-ci sont corrects, une autre fonction va permettre de savoir si la trame mesurée est cohérente en vérifiant les bornes du plus d'éléments constituant la date, possible. (e.g. si l'heure reçue se situe bien entre 0 et 23).

Enfin, si la date a passé tous ces tests, alors on la transforme en timestamp et on modifie le timestamp système avec la fonction *stime()*. Le programme de rafraichissement des LEDs qui se situe dans le dossier */root/horloge/ledPanel* au nom de *heure.cpp* va appeler toute les secondes, la fonction *time()* pour récupérer l'heure système et l'afficher sur le panneau de LED inférieur.

2. Position de l'antenne

Pour une réception optimale des bits l'antenne doit-être positionné à l'horizontale et être normale à la direction de Francfort.

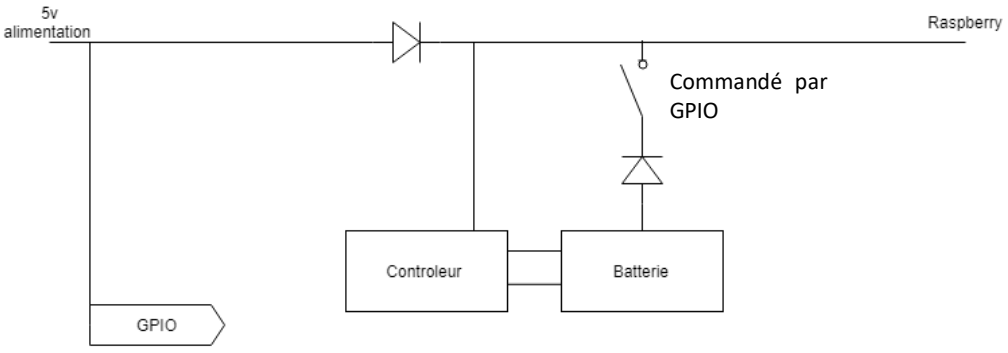


Nos différents tests réalisés nous ont permis de déduire plusieurs critères pour une bonne réception :

- L'antenne doit être placé à distance d'appareils électroniques
- La structure métallique du bâtiment influe sur la qualité de réception
- Une meilleure réception se situe au niveau des fenêtres d'un bâtiment

IV - Gestion de l'extinction brutale

Pour gérer l'extinction brutale nous avons choisi de partir de ce schéma de principe :



Le principe est, quand on débranche l'alimentation de la prise secteur la tension d'entrée du circuit devient plus faible que celle de la batterie, la diode au-dessus de la batterie devient passante et la batterie alimente la raspberry. Une diode est présente pour éviter que la batterie

alimente son contrôleur de charge. Une pin du GPIO est reliée à l'alimentation pour détecter le débranchement lorsqu'une absence de tension est détectée on démarre l'extinction de l'OS Raspbian de la Raspberry Pi. La dernière action que fera la Raspberry sera de déconnecter la batterie d'elle-même, permettant ainsi un allumage de l'OS lors du prochain branchement de la prise électrique sur le secteur. Cette déconnexion est réalisée en écoutant un autre GPIO qui passera de l'état haut à l'état bas, dès que Raspbian est éteint.

1. Choix composant réalisation de la carte électronique

4.1.1. Batterie, contrôleur de batterie

Nous avons choisi une batterie à la tension inférieure à celle de l'alimentation (environ 5.2 V) c'est à dire 4.8v pour permettre le changement d'alimentation commander par la diode.
Concernant la capacité de la batterie nous avons choisi un modèle de 500 mAh ce qui laisse suffisamment de temps pour alimenter la Raspberry durant l'extinction. Nos divers test d'extinction via la commande *halt* nous ont montré que l'extinction durait environ 10 secondes. La capacité de la batterie permet plusieurs coupures de suite sans que la batterie ne soit déchargée. La batterie choisi est de la technologie Nickel-hydrure métallique

Le contrôleur est évidement choisit en fonction des caractéristiques de la batterie c'est-à-dire 4.8v Nickel –hydrure métallique. Le choix s'est porté sur le LT1512. Pour faire fonctionner ce contrôleur de batterie nous avons utilisé l'application typique du composant (Annexe 1). Le transistor 2N2222A est un transistor habituel des circuits à basse puissance car les grandeurs électrique que nous lui soumettons sont plutôt de cet ordre.

4.1.2. Diode, interrupteur commandé

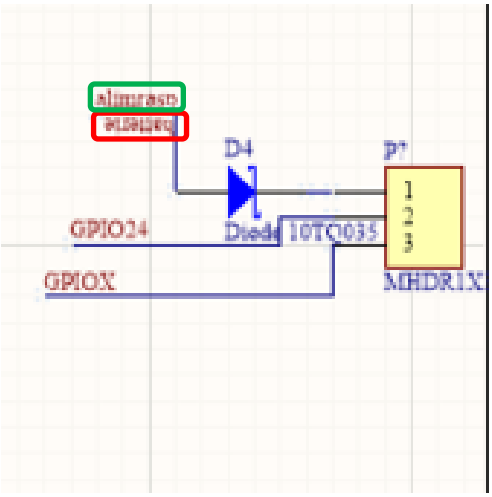
Pour que la commutation d'une alimentation à une autre se réalise rapidement et que la Raspberry ne subisse pas de coupure d'alimentation nous avons opté pour des diodes de type Schottky référence 10TQ035, elles permettent une commutation rapide avec un seuil de tension plus bas également.
Pour couper l'alimentation nous avons malencontreusement réalisé un montage inverseur à transistor NPN. Lorsque la Raspberry subit une coupure de courant, la tension de sortie des pins atteint 0V, or le fil relié au collecteur du transistor et qui alimente la Raspberry retourne 5V alors que nous souhaitons 5V pour une valeur de pin GPIO de 3.3V. Pour corriger l'erreur nous pensions rajouter un deuxième inverseur mais nous n'avons pas eu le temps de refaire le PCB.

4.1.3. Vérification d'absence de tension sur l'alimentation

Nous avons réalisé un point diviseur de tension afin d'envoyer une tension d'environ 3.3V (niveau logique 1) sur la pin GPIO de détection de débranchement secteur. Cette tension provient de l'alimentation du circuit de puissance des LEDs. Nous avons remarqué que lorsque le shield n'était pas relié sur secteur et que l'alimentation provenait du port USB de la Raspberry

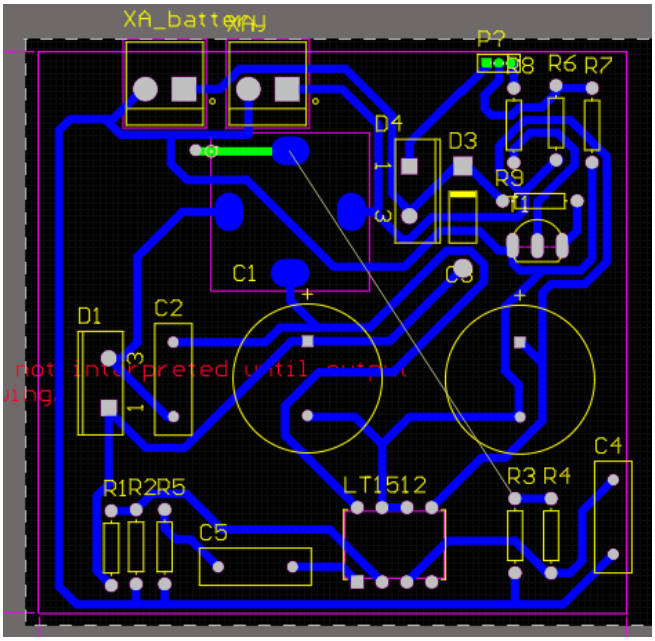
4.1.4. Schematic et PCB

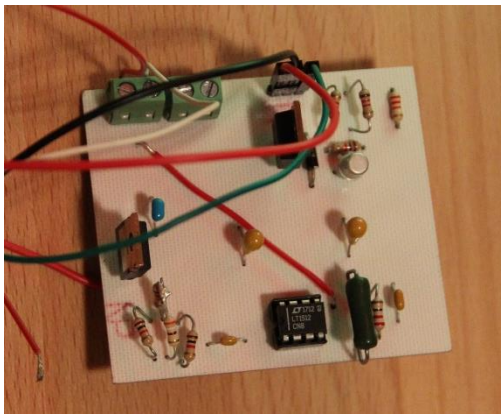
Nous avons réalisé un Schematic sur Altium (Annexes 2) lors de la réalisation du Schematics nous avons commis une erreur, nous avons inversé un netlist. Si on reprend notre schéma de principe on peut voir que l'alimentation de la Rapsberry est reliée d'abord au montage avec transistor hors dans notre schematics nous avions relié l'alimentation de la Raspberry directement à la batterie sans le voir car le netlist « alimrasp» (en vert) est bien présent mais non relié à son homologue présent au collecteur du 2N2222 (ANNEXE 2).

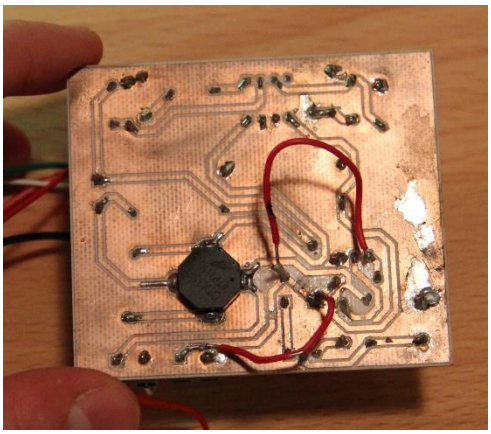


Nous avons vu l'erreur après la fabrication du PCB et avons dû la corriger à la main en faisant passer un straps.

Voici les pistes du PCB que nous avons réalisés :







Le positionnement des composants s'est fait à l'aide de la documentation du LT1512 qui propose un montage limitant les problèmes de compatibilités électromagnétique.

2. Test réalisés et résultats

Le PCB a été testé plusieurs fois. Hormis le problème d'inverseur au niveau du transistor, tout le montage fonctionne et répond à nos attentes. Lorsque la tension d'entrée passe à 0V, la batterie prend correctement le relai. Si la tension 5V en entrée depuis le secteur est présente, la batterie se charge. Le contrôleur de batterie remplie correctement son travail en bloquant la charge lorsque la batterie atteint sa tension maximale. Cependant nous n'avons pas pu l'intégrer à l'horloge à cause du comportement imprévu du montage au niveau du transistor.

CONCLUSION

Notre projet fonctionne et nous avons su répondre à tout le cahier des charges hormis le point sur le contrôle de coupure de courant. Nous avons choisi de ne pas intégrer le PCB à cause de l'erreur que nous avons commise en réalisant malencontreusement un inverseur plutôt qu'un interrupteur. Le cahier des charges aurait pu être complètement validé si nous avions refait un PCB, mais nous manquions de temps. Mais nous pouvons quand même nous satisfaire d'avoir non seulement réussi la partie contrôle de batterie du PCB, mais surtout toute la partie interactive. La réception du DCF77 est aussi une réussite, nous avons pu cerner les conditions d'une bonne réception et pu mettre au point un programme robuste de décodage de l'heure.

Notre projet a été une expérience enrichissante en matière de connaissance et de pratique, nous avons pu apprendre à mieux réaliser nos phases de test notamment pour la partie réception via l'antenne DCF77. Nous avons pu pratiquer de la programmation, du design de circuit électronique, ainsi que du traitement de signal. Ce projet est donc complet sur le plan technique.

BIBLIOGRAPHIE

Ma station météo et Ma domotique
http://s159401799.onlinehome.fr/600HorlogeAtomiqueDCF77.html

Datasheet du LT1512
https://docs-emea.rs-online.com/webdocs/1364/0900766b813648fa.pdf

ANNEXES

ANNEXE I :typical application LT1512

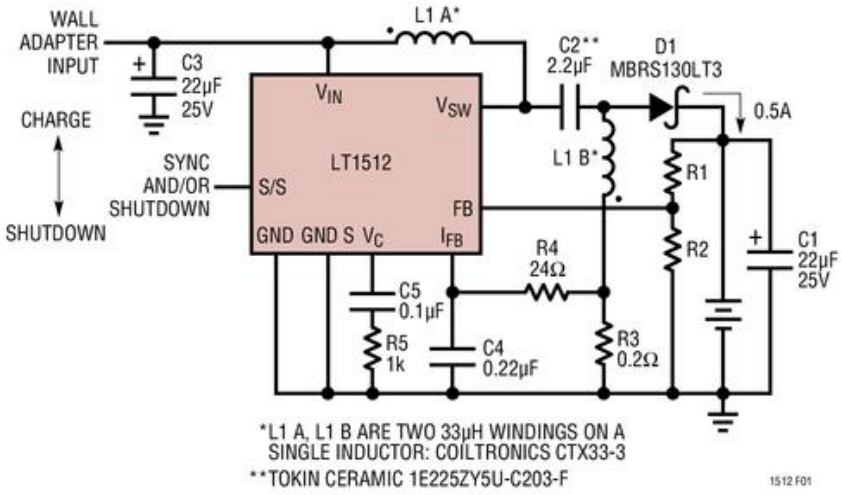


Figure 1. SEPIC Charger with 0.5A Output Current

ANNEXE II : Schematics