CLERMONT AUVERGNE INP Isima

LIMOS

Rapport d'élèves ingénieurs

Projet de 2e année

Filière : Informatique des systèmes interactifs pour l'embarqué, la robotique et le virtuel

Balise sonore

Présenté par : Baptiste Daniel-Lamazière et Nicolas Laval

Responsable ISIMA : Bastien Doreau — 03 mars 2022

Responsable entreprise : Jean-Marie Favreau — 60h de travail

Campus des Cézeaux : 1 rue de la Chébarde. TSA 60125. 63175 Aubière CEDEX

Remerciements

Nous souhaitons remercier notre tuteur, Jean-Marie Favreau, ainsi que Jérémy Kalsron pour leur précieuse aide du début du projet jusqu'à la relecture de ce rapport. Nous tenons aussi à remercier les membres de l'association Les Petits Débrouillards grâce à qui nous avons pu nous retrouver pour travailler et qui ont initialement proposé ce projet. En particulier, nous voulons remercier Samuel, Colas et Noa qui ont suivi nos réunions et nous ont accompagnés durant les moments les plus critiques de notre projet.

Nous tenons à remercier notre référent ISIMA*, Bastien Doreau, qui a assisté à notre revue de projet en compagnie d'Yves-Jean Daniel et de Mamadou Kanté que nous remercions également.

Nous remercions Murielle Mouzat pour ses conseils lors du cours de Communication et son implication auprès de tous les élèves dans la préparation des soutenances de projets.

Nous témoignons aussi toute notre reconnaissance à Michel Cheminat pour la qualité de ses cours et particulièrement celui de Transmission de Données Sécurisées qui nous a été très utile dans ce projet. Nous en profitons pour remercier également l'ensemble de nos professeurs, de l'équipe pédagogique de l'ISIMA et des membres du LIMOS*.

Nous adressons un grand merci à Frédéric Meignan pour son assistance et le prêt de matériel que nous avons beaucoup utilisé lors de ce projet.

Résumé

Les personnes atteintes d'une déficience visuelle ont beaucoup de difficultés pour se déplacer. C'est notamment le cas pour la traversée des passages piétons. Pour aider ces personnes à traverser, certains feux piétons sont équipés d'une balise sonore afin d'émettre des sons pour aider la traversée. Les personnes malvoyantes ont souvent une télécommande afin d'activer les balises au moment opportun.

Les solutions déjà présentes sur le marché sont fiables mais sont chères par rapport à la faible complexité du système. Nous souhaitons réaliser une balise utilisable dans les mêmes situations et pouvant être utilisée par des particuliers ou des associations afin d'y trouver d'autres utilités.

Le but de ce projet est de réaliser une telle balise. Le choix des composants est fait pour répondre à un petit budget afin que cette balise soit réalisable par tout le monde. En effet, cette balise a pour but d'être open source pour aider un maximum de personnes et pour différents usages si besoin.
Nous allons utiliser une Raspberry Pi Pico ainsi qu'un récepteur Texas Instrument CC1101 pour réaliser la base du projet.

Mots-clés : Raspberry Pi Pico, CC1101, STEP-HEAR, Arduino, Communication Radio

Abstract

Sight impaired people have much greater difficulties to move about. It is particularly the case for the crossing of a street. To help these people to cross the street, some traffic lights are equipped with a sound beacon in order to make sounds to help with the crossing. The sight impaired often carry around a remote that allows them to activate the beacons at the timely moment.

Reliable solutions are already available on the market but they are expensive considering the low complexity of these systems. We intend to realize a beacon that could be used in the same situations by individuals or non-profit organizations so that they can find new functions.

The goal of this project is to realise such a beacon. The choice of the components is done to meet a limited budget so that anyone would be able to make this beacon. Indeed, this beacon intend to be open-source in order to help as many people as possible in various functions if needed.
We are going to use a Raspberry Pi Pico as well as a receiver CC1101 from Texas Instrument in order to realize the core of this project.

Key-words : Raspberry Pi Pico, CC1101, STEP-HEAR, Arduino, Radio Communication

Sommaire

Liste des Figures

Références bibliographiques

[Tex13] TexasInstruments. "CC1101 Low-power Sub-1 GHz wireless transceiver". In : (2013).

[Tex14] TexasInstruments. "CC Debugger User's Guide". In : (2014).

[AFN15] AFNOR. "Dispositifs répétiteurs de feux de circulation à l'usage des personnes aveugles ou malvoyantes". In : (2015).

[Pi21] Raspberry Pi. "Raspberry Pi Pico C/C++ SDK". In : (2021).

[Ras21] RaspberryPi. "Raspberry Pi Pico Datasheet". In : (2021).

1 Introduction

Notre projet consiste en la conception d'une balise sonore capable de reproduire le fonctionnement des balises installées sur les feux de circulation au niveau des passages piétons ainsi que le fonctionnement des balises STEP-HEAR disponibles dans le commerce.

Pour réaliser ce projet, il y a deux grandes étapes. La première est de comprendre la norme de communication entre les télécommandes des malvoyants et les balises audio principalement installées sur les feux piétons. La deuxième grosse étape est la communication en SPI* entre le micro-contrôleur et l'antenne. Dans un premier temps, nous avons étudié la norme proposée par l'AFNOR*. [AFN15] Cette norme est assez compliquée selon nous, ce qui nous handicape particulièrement.

En parallèle, nous avons essayé d'utiliser le matériel à disposition. Le micro-contrôleur étant récent sur le marché, il n'a pas été évident de trouver une solution valable pour compiler un programme dessus.

Nous allons donc principalement voir comment nous nous sommes adaptés à la norme de communication et comment nous avons réussi à utiliser le micro-contrôleur dans les meilleures conditions.

1.1 Balise sonore et feux piétons

1.2 Principe de conception

Le but de ce projet est de réaliser une balise sonore. Les possibilités d'extensions sont presque infinies mais pendant le temps qui nous est donné, nous allons nous consacrer à la base de la balise. Nous allons utiliser un micro-contrôleur et un émetteur/récepteur afin d'écouter et de décoder les trams qu'envoient les télécommandes dont sont munis les malvoyants.

Pour se faire, il faut comprendre comment fonctionne le micro-contrôleur Raspberry Pi Pico et l'émetteur CC1101. La communication entre les deux composants est de type SPI.

D'un autre côté, nous devons réussir à interpréter la trame envoyée par les télécommandes afin de pouvoir la décoder après que le récepteur l'ai envoyée au micro-contrôleur.

Une fois que la communication SPI sera mise en place et que le décodage de trames sera opérationnel, le projet aura une base solide pour voir des extensions et des applications. Nous pouvons imaginer dans un premier temps l'ajout d'un haut parleur afin d'émettre des sons, une possibilité d'écouter plusieurs trames différentes pour jouer différents sons. Il est aussi possible d'imaginer construire une télécommande qui soit un peu plus modulable que celles disponibles sur le marché actuellement. Autant d'extensions possibles qu'il n'y a de personnes pour y penser. Malheureusement, ce projet est délimité dans le temps et nous ne pouvons qu'en imaginer les suites possibles.

1.3 Contraintes du projet

Les conditions du projet ont été prédéfinies avec nos tuteurs pendant nos premières rencontres. Cette étude a pour but d'aider le grand public. Il faut donc que ce coût final ne soit pas trop élevé. Il est préférable que la balise ne soit pas énergivore pour qu'elle puisse fonctionner avec une petite batterie pour être utilisable même dans des zones sans électricité. Quand le projet sera fini, il devrait être open source donc il faut que la balise soit réalisable sans trop de problème par plus ou moins toute personne qui souhaite avoir une telle balise sonore.

1.4 Organisation du travail

Nous avons planifié les tâches à effectuer dans le cadre du projet à l'aide d'un diagramme de Gantt (Figure 1). Initialement, nous prévoyions que certaines tâches seraient plus faciles à exécuter qu'elles ne l'ont réellement été. De plus, notre diagramme de Gantt prévisionnel ne prenait pas en compte le changement d'approche que nous avons effectué lors de l'écriture du driver SPI pour notre composant principal, ou encore que l'analyse du protocole de communication nous prendrait autant de temps.

FIGURE 1 – Diagramme de Gantt prévisionnel

Bien évidemment, nous n'avons pas pu exécuter l'enchaînement des tâches du projets exactement comme nous le souhaitions. (Figure 2) Ainsi, dès les premiers tests de développement avec MicroPython nous avons rencontré des difficultés qui nous ont ralentis. Les tâches pour lesquelles nous avons passé beaucoup plus de temps que prévu sont l'analyse du protocole de radio-communication, le développement de la liaison SPI en C/C++. Certains de ces obstacles nous ont conduit à ne pas effectuer certaines tâches prévues initialement comme l'implémentation du son sur la Pico. De plus, nous avons abandonné certaines tâches en raison de leur complexité comme le développement du pilote SPI avec le SDK* C/C++.

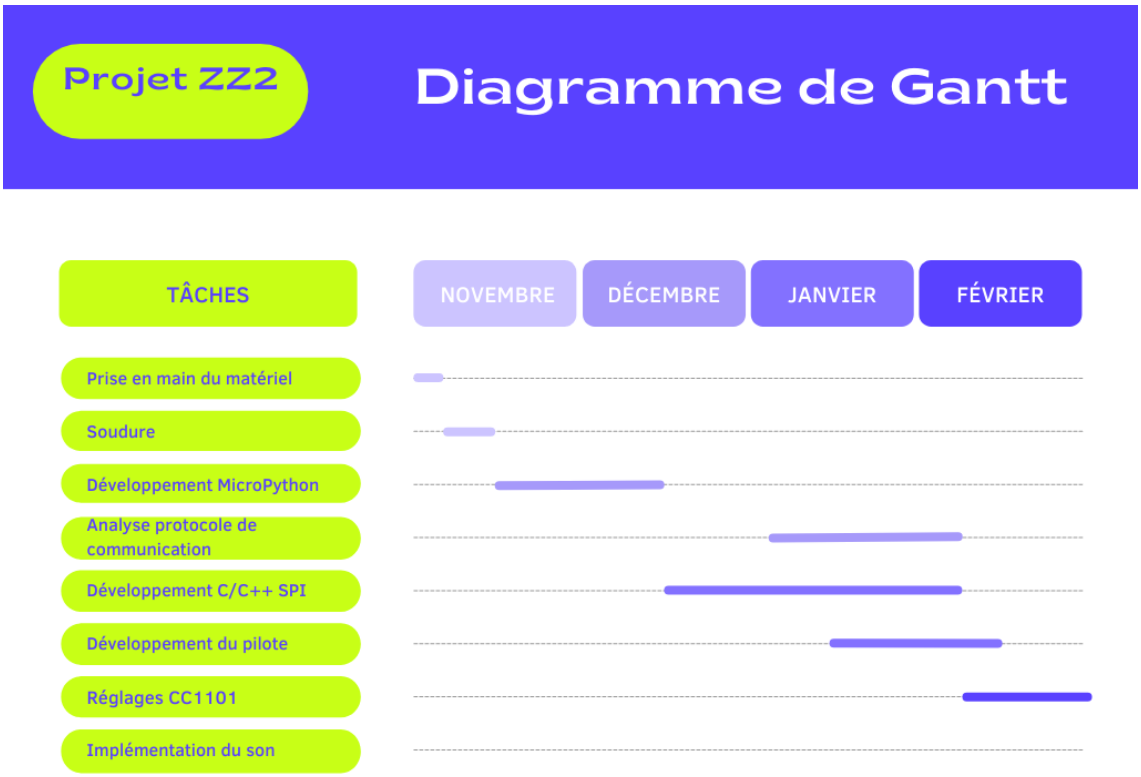

FIGURE 2 – Diagramme de Gantt réel

Il faut aussi préciser qu'afin de réaliser certaines tâches nous avons essayé de paralléliser notre travail. Ainsi, lors de la phase de réalisation du driver SPI, nous nous étions réparti le travail entre une partie sur le code en Arduino et l'autre sur le SDK C/C++ de Raspberry Pi Pico.

Nous avions aussi depuis le début du projet, un dépôt Google Drive pour toute la documentation, ce qui nous permettait de retrouver et de partager des documents rapidement. Par la suite, nous avons aussi utilisé un document en ligne qui regroupait le résumé de nos réunions, les liens vers les sites ou les documents qui nous intéressaient ainsi que des indications que nous échangions. Pour regrouper toutes ces informations nous avons utilisé la plateforme en ligne Hedgedoc.

1.5 Materiel à disposition

Maintenant que nous avons exposé ce que nous avons eu à faire, il est préférable de comprendre le matériel que nous avons utilisé. On peut diviser l'ensemble de ce matériel entre les éléments qui composent la balise que nous traiterons en premier et les éléments qui nous ont servi lors de l'étude du protocole de communication et du sujet de manière générale que nous exposerons ensuite.

1.5.1 Raspberry Pi Pico

Le microcontrôleur est le cœur de la balise, nous avons à disposition une Raspberry Pi Pico cité ultérieurement. La Pico est montée autour d'une puce RP2040 accessible par 40 broches. (Figure 3) Le choix de cette carte est à la fois économique, le prix réduit de la Pico (seulement 4€) permet de répondre aux besoins d'un budget réduit pour le projet, et pratique, la Pico est une carte développée par Raspberry Pi cela permet donc d'avoir accès à une documentation très complète sur cette carte et sur la façon de la programmer [Ras21]. Néanmoins, la Pico reste malgré tout une carte récente et atypique parmi les autres Raspberry Pi, par conséquent il existe moins de documentation pour la Pico que pour les autres Raspberry Pi et le développement reste assez distant de ce qu'on peut trouver sur Raspberry Pi.

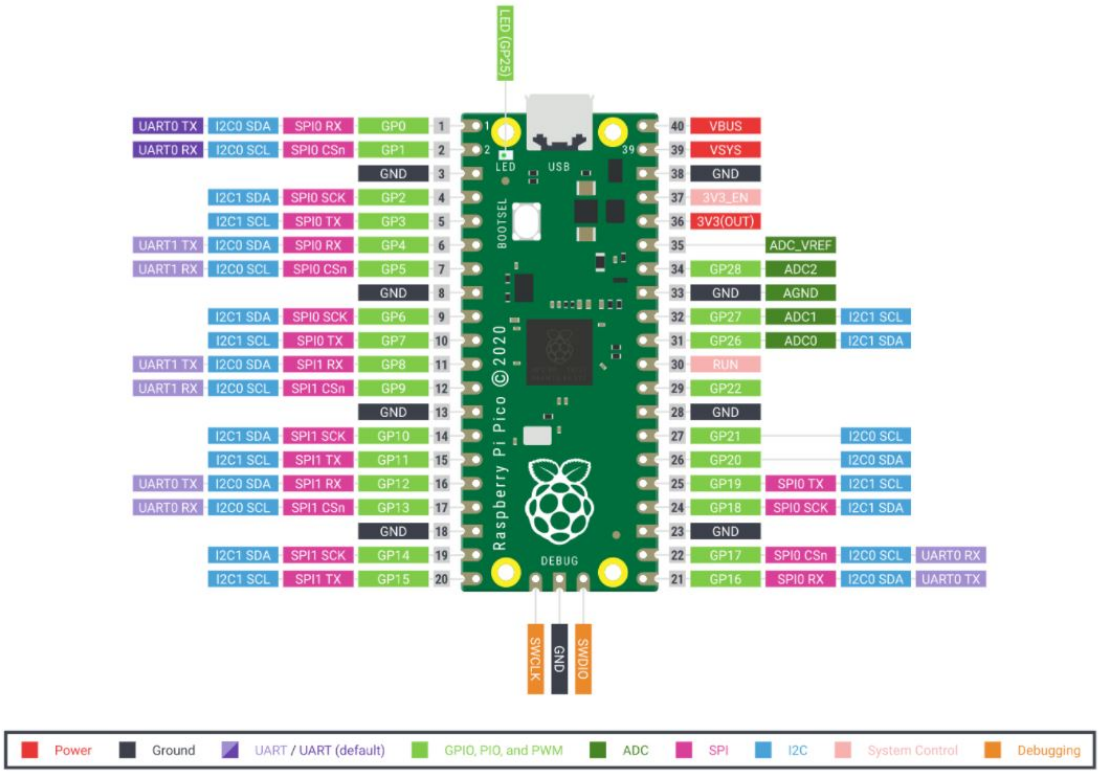

FIGURE 3 – Broches de la carte Raspberry Pi Pico

1.5.2 CC1101

La communication entre la télécommande est possible grâce à un émetteur du côté de la télécommande, et un récepteur du côté de la balise sonore. La CC1101 (Figure 4) de chez Texas Instrument permet la réception et l'émission sur les bandes ISM* et SDR* autour des fréquences 315/433/868/915 MHz. [Tex13] Nous écouterons principalement sur la fréquence 868.3 MHz sur laquelle la norme de communication que nous utilisons. Le CC1101 présente aussi l'avantage d'être à bas coût ce qui est important dans le cadre de notre projet. Lors de la mise en place du CC1101 il nous a suffit de souder l'antenne ainsi que les câbles destinés à assurer la connexion avec la Pico. Il faut bien veiller à alimenter le composant avec la broche de tension à 3,3V et non celle à 5V qui risquerait d'endommager la CC1101. (Figure 5)

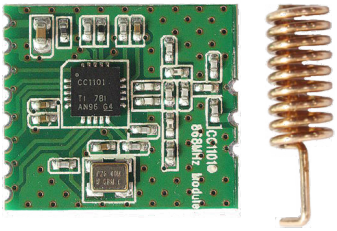

FIGURE 4 – Récepteur CC1101 avant soudure

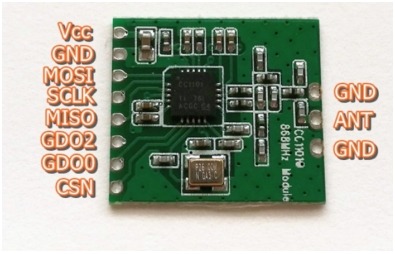

FIGURE 5 – Récepteur CC1101 avec les broches en conséquence

Comme on le voit sur la Figure 5, le CC1101 dispose de 2 broche de sortie numérique (Digital Output Pins) appelées GDO0 et GDO2, nous verrons par la suite que nous n'avons pas besoin de brancher ces sorties numériques à la Pico dans le cadre de l'écoute du signal de télécommande.

Le CC1101 possède de nombreux registres permettant de le programmer pour l'écoute d'un signal particulier. Parmi ces registres, l'un des plus utiles est le registre MARCSTATE qui donne l'état dans lequel se trouve le CC1101, en effet le CC1101 est une machine à état finis et MARCSTATE permet de savoir si le composant est en mode de réception, de transmission ou en mode inactif. Ce mode inactif, appelé IDLE peut être accédé via le registre SIDLE qui met le CC1101 dans cet état et le fait sortir du mode dans lequel il se trouvait auparavant. Les registres SYNC0 et SYNC1 sont les registres définissant le mot de synchronisation attendu par le CC1101, le SYNC1 stocke les bits de poids fort et le SYNC0 stocke les bits de poids faible. Il existe aussi de nombreux registres pour contrôler la fréquence d'écoute du CC1101, FREQ0, 1 et 2 sont les registres définissant la fréquence d'écoute du CC1101. De nombreux autres registres permettent de configurer le CC1101 comme on le souhaite. Pour notre part nous avons eu besoin de maîtriser les différents rôles de ces registres pour écrire le driver. Finalement, nous allons voir que certaines librairies prennent totalement en compte la modification de ces registres lors de la configuration du CC1101.

1.5.3 Balise Sonore STEP-HEAR

Pour pouvoir réaliser une balise sonore, il est préférable d'en avoir une. Nous en avons une du constructeur STEP-HEAR ainsi qu'une télécommande fournie avec. (Figure 6) La balise permet d'enregistrer trois sons différents qui sont jouables grâce à trois boutons différents sur la télécommande. Cette télécommande nous sera très utile par la suite pour écouter la trame. Il est à noter que cette balise est légèrement différente des bornes sur les feux de trafic au niveau des passages piétons, elle reçoit un signal différent et permet plus d'actions différentes que les balises trouvées sur les passages piétons.

FIGURE 6 – La balise sonore STEP-HEAR SH200 avec la télécommande SH220 associée

FIGURE 7 – L'interface de l'application mobile STEP-HEAR

En effet, la balise STEP-HEAR ne peut pas être activée par tous les types de télécommandes. Par exemple, la télécommande de Jérémy Kalsron, un de nos tuteurs qui nous a aidé tout au long du projet, ne permettait pas d'activer la balise STEP-HEAR. En revanche, cette balise, lorsqu'elle est activée par une télécommande conforme, peut émettre trois sons différents au lieu d'un seul pour les balises sonores classiques. De plus, cette balise peut aussi être activée à l'aide d'un téléphone mobile via bluetooth et une application STEP-HEAR (Figure 7 dédiée. Nous avons pu essayer cette application avec la balise STEP-HEAR fournie, mais malheureusement nous n'avons jamais pu tester son fonctionnement avec une balise "classique".L'application permet aussi de localiser les balises déjà connues sur un plan de ville, de donner la direction dans laquelle aller pour retrouver une balise ou de détecter des balises STEP-HEAR à proximité.

1.5.4 RTL-SDR

La RTL-SDR est un équipement de radio logicielle, il est constitué d'une clé USB* permettant de programmer l'écoute radio et d'une antenne. (Figure 8) Nous utilisons cet équipement afin d'écouter précisément les trames échangées entre la télécommande et la borne. Pour ce faire, nous devons d'abord paramétrer la RTL-SDR pour une écoute sur la fréquence 868.3 MHz de la bande ISM (bande industrielle,scientifique et médicale) utilisée par le protocole des bornes déjà existantes. La bande ISM est une une plage de fréquences sur laquelle les communications sont réglementées. En effet l'émission par un même appareil sur une bande ISM est limitée à 1% du temps, ce problème est normalement déjà pris en compte par les fournisseurs de télécommandes. L'exploitation des données écoutées avec la SDR(Software Defined Radio) se fait via le logiciel Universal Radio Hacker. Universal Radio Hacker permet de visualiser les trames écoutées de façon graphique mais aussi de décoder les bits reçu par l'antenne.

Le choix de la RTL-SDR et du logiciel Universal Radio Hacker a été justifié par le fait que nous connaissions déjà grâce au cours de Transmission de Données Sécurisée de M.Cheminat. Lors de ce cours nous avions écouté un signal de clé de voiture à distance et avions utilisé la fonction de décodage du logiciel Universal Radio Hacker. La séance de Travaux Pratiques de ce cours nous a été très utile pour appréhender l'écoute des télécommandes et le décodage des trames envoyées.

FIGURE 8 – Clé RTL SDR utilisée pour sniffer
la fréquence 868.3 MHz

1.5.5 Arduino Nano

L'Arduino Nano est une carte Arduino de format similaire à celui de la Pico, la Nano possède un micro-contrôleur ATMega328 et 22 broches dont 8 broches analogiques et 14 broches digitales. Bien que très semblable à la Pico sur de nombreux points, elle coûte un peu plus cher (20 €) et c'est pour cela qu'elle n'a pas été préférée à la Pico pour ce projet. Nous nous sommes servis de cette carte pour tester des programmes avec l'IDE Arduino. Les premiers programmes testés étaient très simples, par exemple faire clignoter une LED. L'IDE Arduino est compatible avec la Pico et étant donné que certaines librairies pour la discussion SPI avec CC1101 existaient déjà pour Arduino nous avons fini par utiliser ces solutions.

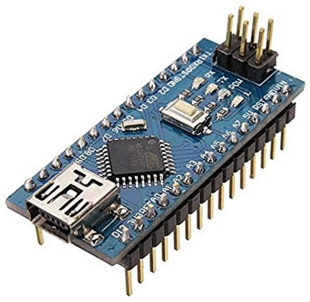

FIGURE 9 – La puce Arduino Nano

1.5.6 CC Debugger

Nous avons eu l'opportunité d'avoir un debugger afin de vérifier si les récepteurs CC1101 fonctionnaient correctement, notamment après les avoir soudés. Le principe est simple. Il suffit de regarder dans la documentation [Tex14] comment brancher les pins du récepteur CC1101 (Figure 10). Ensuite il faut installer le logiciel SmartRF Studio 7 via le site de Texas Instrument puis de brancher le debugger USB à l'ordinateur. Ceci étant fait, le plus dur est passé. Le logiciel détecte quel est le type de CC branché. En effet ce debugger fonctionne aussi bien pour les CC2510 que les CC1101 ou autres. Une fois que le CC1101 est détecté, il est possible d'écouter un signal avec le mode RX* ou transmettre un signal avec le mode TX*. (Figure 11)

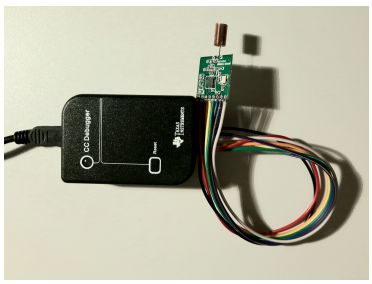

FIGURE 10 – Récepteur CC1101 branchée au CC Debugger

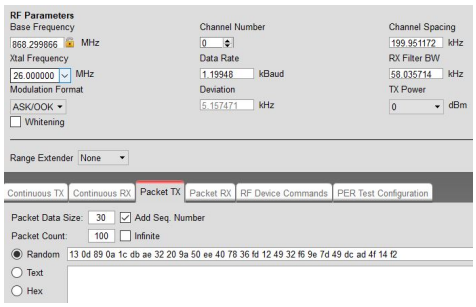

FIGURE 11 – Capture d'écran du logiciel SmartRF

2 Réalisation du projet

Au début du projet nous nous sommes concentrés sur la lecture des fiches techniques et de la norme mise en place pour la communication que l'on étudie. Ensuite, nous nous sommes interrogés sur la compilation vers la Pico. Nous avons essayé Micropython, un langage pour les micro-contrôleurs en python. Par la suite, nous nous sommes intéressés au SDK puis à la compilation via l'IDE Arduino avec des librairies adaptées pour compiler vers une Raspberry Pi Pico avec la possibilité de communiquer en SPI avec notre récepteur.

2.1 Premières rencontres

Lors de la première réunion nous avons rencontré plusieurs chercheurs au LIMOS, dont certains étaient aussi bénévoles de l'association Les Petits Débrouillards. Cette réunion a consisté en une explication du projet, une présentation du matériel mis à disposition et une discussion sur les objectifs du projet et les possibilités d'extensions de celui-ci. Nous avons par la suite reconduit ces réunions avec moins de participants au rythme d'une réunion toutes les deux semaines environ. Nous avons aussi rencontré des membres de l'association Les Petits Débrouillards dans le FabLab de l'association : Le DébrouilloLab.

Nous avons appris au cours de ces réunions que l'un des buts après le développement final de la balise open-source serait le développement d'une balise autonome en énergie. Cette balise autonome, ne nécessiterait pas d'être raccordée au secteur pour fonctionner, nous parlerons de cette extension du projet plus loin dans ce rapport. Une autre extension possible serait la mise en place d'exposition sonores à l'aide de balises sonores à bas-prix mais dont l'émission du son serait plus travaillée.

2.2 Tests du matériel

Pour commencer, nous avons essayé de faire fonctionner les différents items du matériel que nous avions à disposition. Nous avons activé la balise STEP-HEAR à distance, nous n'avons toutefois pas réussi à enregistrer un nouveau son audible pour que la balise émette un son différent que celui par défaut. Nous avons aussi branché sur nos ordinateurs la Pico, dans un cas, l'ordinateur ne détectait pas le port série de communication utilisé par la Pico pour communiquer avec l'ordinateur et permettre plus tard la cross-compilation.

Nous avons aussi, plus tard dans le projet, testé le bon fonctionnement du composant CC1101 avec le CC debugger de Texas Instruments, nous avons réalisé ce test du composant au même moment que l'écoute du protocole avec la radio-logicielle, afin de comparer les deux signaux entendus. Nous avons pu constater que CC1101 "entendait" la même chose que la RTL-SDR, ce qui a permis de confirmer le bon état du composant.

2.3 Soudure

Lors de la mise en place du projet, nous avons d'abord soudé les broches de la Pico, nous avons réalisé cette étape en soudant les broches à l'envers. (Figure 12) Ce choix a eu pour conséquence de rendre les broches lisibles directement sur la Pico, ce qui facilitait grandement le câblage des composants à la Pico, comme le piézo-buzzer lors des tests en MicroPython ou le CC1101 par la suite.

En revanche, le fait de souder les broches à l'envers rend difficilement accessible le bouton Bootsel situé sur l'autre face de la Pico et donc du côté de la whiteboard. Or le bouton Bootsel est très souvent utilisé lors du développement sur Pico, à chaque fois que l'on souhaite téléverser une nouvelle version du programme il faut en effet maintenir le bouton Bootsel enfoncé lorsqu'on branche la Pico à un ordinateur. Cette soudure nous a donc permis de gagner du temps au début du projet ainsi qu'à chaque nouveau câblage mais nous handicape à chaque téléversement sur la carte. Il reste toutefois facile de Reset la carte à chaque téléversement en appuyant sur le bouton Bootsel avant de brancher la Pico sur la whiteboard de sorte que le bouton Bootsel est facilement accessible. Pour la deuxième Pico que nous avons a disposition, nous avons fait la soudure dans le bon sens. (Figure 13)

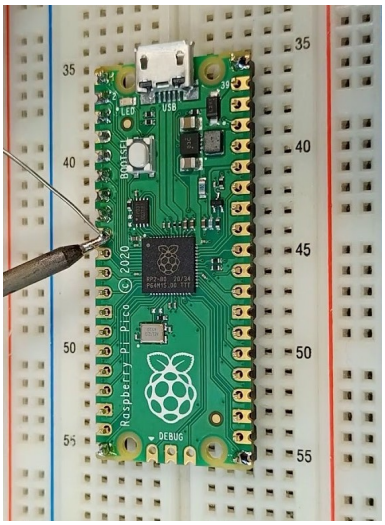

FIGURE 12 – Pico soudée correctement

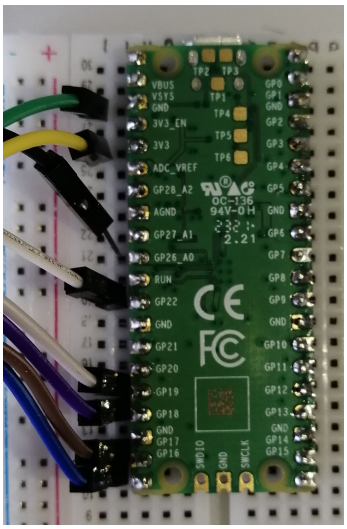

FIGURE 13 – Pico mal soudée

2.4 Normes

Cette norme autorise la fréquence 868.3 MHz sur la bande ISM. La trame qui explicite la communication est composée d'un Préambule, d'une Synchronisation et d'un Code où se trouvent les données à interpréter [AFN15]. Ces trois parties de la trame sont malheureusement sur des périodes d'échantillonage différentes. En effet, si l'on se penche sur le préambule, un bit est codé sur 415µs tandis que la partie des données est codée sur 500µs par bit. C'est sans parler de la synchronisation qui a deux périodes différentes en 1665µs soit deux bits de 625µs et un bit de 415µs. Le Préambule et la Synchronisation sont codés en demi-bits comme le codage Manchester. (Figure 14) C'est aussi le cas des données, elles sont codées par demi-bits. Si les deux demi-bits sont identiques (00 ou 11) alors le bit est 1, à l'inverse, si les deux demi-bits sont opposés (10 ou 01) alors le bit est 0. Le code de données qui est inscrit dans la norme donne 00A833 en hexadécimal. Sauf que les caractères de poids faibles sont envoyés en premier. Cela donne donc 338A00 et les 3 en binaires ne sont pas 0011 comme attendu mais bien 1100 vu que les caractères de poids faibles sont en premier.

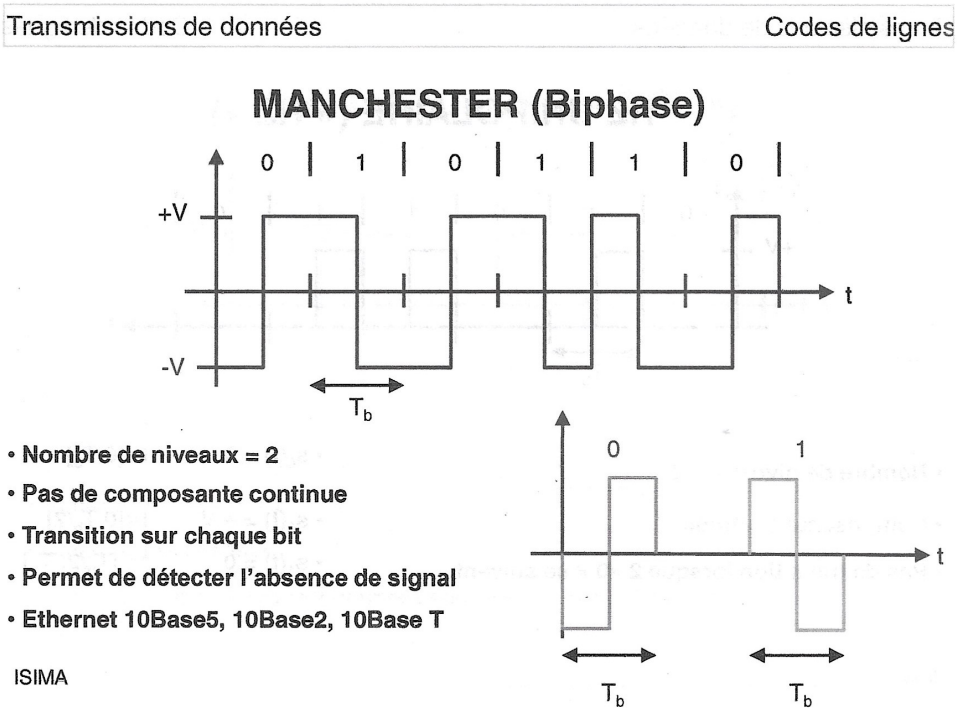

FIGURE 14 – Définition du code Manchester d'après le cours de Transmission de Données Sécurisée de M. Cheminat

2.5 Écoute du signal

Après l'analyse de la trame, il est venu le moment d'écouter la trame envoyée par une télécommande, afin de voir si cela correspondait à la norme. Pour cela, nous avons utilisé une radio logicielle prêtée par l'ISIMA. Nous nous sommes inspirés d'une séance de Travaux Pratiques fait en cours (en Transmission de Données Sécurisée avec M. Cheminat). Ainsi avec une clé USB RTL-SDR et le logiciel Universal Radio Hacker, nous avons pu écouter la fréquence 868.3 MHz. Les trames ont défilé à chaque fois que l'on a appuyé sur un bouton de la télécommande. Après quelques essais et quelques réglages nous avons pu observer les trames qui correspondaient à la norme. Afin d'écouter correctement la trame, il nous a fallu fixer le paramètre de nombre d'échantillons pour un symbole (équivalent au baudrate) à 300. Bien que ces réglages nous permettent de lire correctement les bits bruts envoyés par la télécommande, cela ne nous permet pas de décoder le message envoyé par la télécommande.

Grâce à Universal Radio Hacker nous avons pu retrouver ce que la norme indiquait, à savoir un changement de la période d'échantillonnage au cours du signal de préambule. En effet, la durée affichée par le logiciel pour les différents bits au cours du message change. De plus, nous avons remarqué des différences de signal entre nos deux télécommandes : celle qui permet d'activer la borne audio SH200 de STEP-HEAR, envoie un premier mot de synchronisation non conforme à ce qui est attendu dans la norme. Toutefois, le second mot de synchronisation envoyé est conforme à la norme. L'autre télécommande, plus ancienne, semble mieux respecter la norme, mais la durée d'un bit n'est pas tout à fait celle à laquelle on s'attend en se fiant uniquement à la norme.

Lors de l'écoute des deux télécommandes on remarque aussi une différence de signal entre les deux télécommandes. Les deux télécommandes fonctionnent toutes deux sur les balises sonores installées en ville mais seule la télécommande fournie avec la balise STEP-HEAR permet d'activer la balise. La vieille télécommande (qui ne fonctionne qu'avec les balises de ville) envoie un message qui correspond quasi-exactement à ce qu'on attend après avoir lu la norme concernant le protocole. Sur la figure 15, on peut distinguer deux signaux différents via les deux amplitudes différentes. La vieille télécommande (signal avec l'amplitude faible sur la Figure 15) envoie le même message en boucle si on maintient le bouton appuyé. Ce message correspond au préambule et au mot de synchronisation. Tandis que la télécommande plus récente envoie des trames séparées et distinctes (avec une amplitude plus grande sur la Figure 15) afin de ne pas saturer la bande ISM.

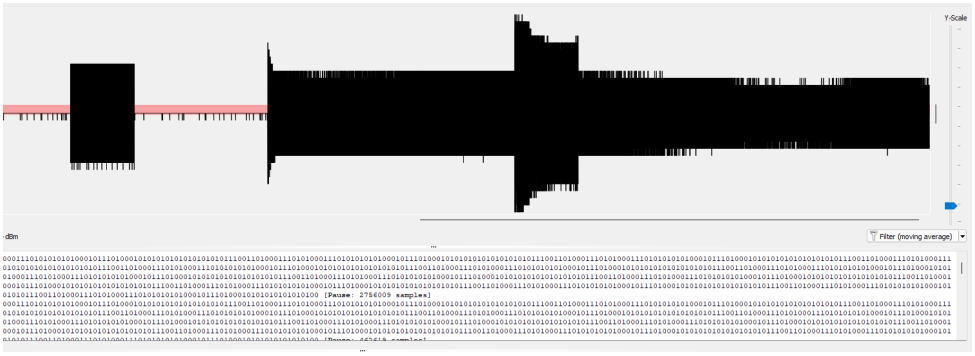

FIGURE 15 – Signaux reçus simultanément
de deux télécommandes différentes

La télécommande plus récente envoie quant à elle le premier front descendant du premier bit du préambule ainsi que la suite du préambule. Puis, surprenamment, elle envoie un signal aussi long que le mot de synchronisation qui ne correspond pas à ce que nous lisons dans la norme (en rouge sur la Figure 16). Le message se poursuit avec l'envoie d'un second préambule et d'un mot de synchronisation conforme à ce qu'on lit dans la norme. Le même enchaînement de signaux se répète un nombre fini de fois, le même signal n'est envoyé en continu lors de l'appui sur le bouton. Étant donné que la bande 868.3 Mhz est réglementée avec une émission maximale autorisée pendant 1% du temps à un même appareil, la deuxième télécommande semble mieux se plier à cette exigence.

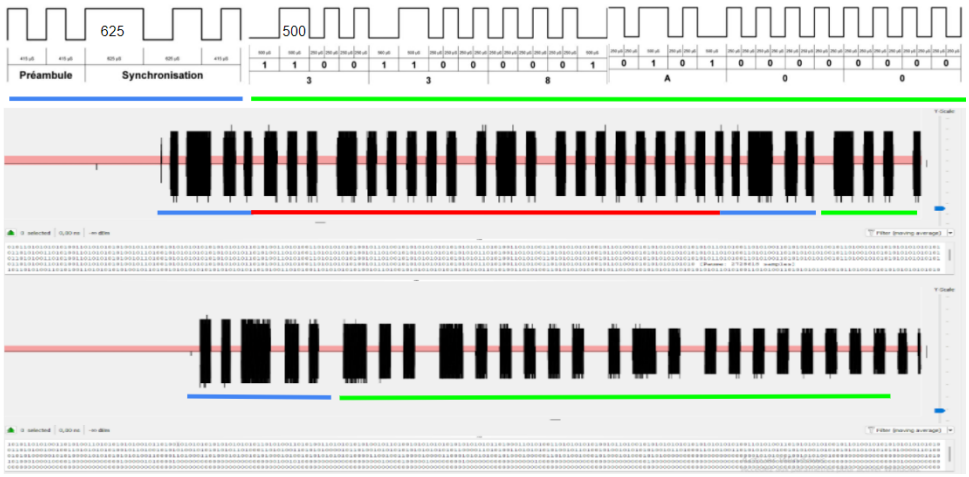

FIGURE 16 – Comparaison du protocole défini par la norme
et de ceux reçus à l'activation avec chacune des deux
télécommandes

2.6 Compiler pour la Pico

2.6.1 MicroPython

Nous avons d'abord eu quelques difficultés à mettre en place la Pico. Le problème principal étant que l'ordinateur que nous utilisions ne détectait pas la Pico et son port série. Ce petit obstacle peut être surmonté en allant dans le gestionnaire de périphériques et en supprimant puis en réinstallant les pilotes logiciels pour les ports COM que la Pico utilise. Bien que facile à résoudre, il faudra veiller à bien documenter ce problème avant de finaliser le projet pour éviter que cela détourne un public néophyte dès le début de la réalisation de la borne open-source.

Étant donné que la Pico supporte MicroPython, un langage très facile d'accès et simplifié, nous avons d'abord jugé que le pilote de CC1101 pouvait être codé en MicroPython ce qui aurait été une solution plus simple et plus rapide que de le coder en C ou C++. Il fallait donc comprendre comment fonctionnait cette version améliorée du Python destinée aux micro-contrôleurs. Dans cette partie du projet nous avons utilisé l'IDE Thonny qui permet de compiler et téléverser le code en MicroPython sur la Pico.

Pour ce faire nous avons codé des codes basiques, nous permettant de tester des fonctions simples de MicroPython. Grâce à cela, nous avons constaté que la Pico fonctionnait bien. Des codes basiques ont bien fonctionné comme le code Blink pour allumer la diode présente sur la Pico (Figure 17), ou encore allumer une diode via un bouton poussoir et pour finir nous avons utilisé un buzzer piézoélectrique pour émettre du son avec la Pico.

Une fois ces codes basiques testés, nous devions passer à la réalisation de la liaison SPI avec le CC1101. Il existe déjà des fonctions pour SPI dans la librairie "machine" de MicroPython. Malheureusement l'inconvénient principal du Python est son manque de proximité avec le hardware ; si conséquent, si établir une liaison SPI avec le CC1101 s'est avéré fort simple, il nous restait encore à gérer les nombreux registres du CC1101. De plus, le MicroPython est encore très peu documenté et nous n'avions accès à aucun code de pilote en MicroPython pour un composant similaire au CC1101. Nous avons donc abandonné en faveur d'un langage de programmation bien mieux documenté et dont nous pouvions trouver des exemples de pilotes pour des composants proches du CC1101. Nous avons par conséquent opté pour le C et le C++.

blinkinled.py

```
1  from machine import Pin, Timer
2
3  led = Pin(25, Pin.OUT)
4  tim = Timer()
5  def tick(timer):
6      global led
7      led.toggle()
8
9  tim.init(freq=2.5, mode=Timer.PERIODIC, callback=tick)
```

FIGURE 17 – Exemple d'un code faisant clignoter une LED en MicroPython avec l'IDE Thonny

2.6.2 SDK C/C++

Nous nous sommes renseignés sur le SDK pour Pico [Pi21], cela nous permettrait une cross compilation efficace entre notre ordinateur et la Pico. Nous avons d'abord essayé de reprendre un driver pour CC1101 sur Raspberry Pi, ce driver trouvé sur GitHub* étant très complet nous souhaitions reprendre les mêmes fonctions qui permettaient notamment une écoute suffisamment fine pour décoder la trame envoyée par la télécommande. (Figure 18) Néanmoins ce driver ayant été conçu pour Raspberry Pi ses créateurs l'avaient codé à l'aide de la librairie WiringPi qui, bien que extrêmement performante sur Raspberry Pi ne fonctionne pas sur d'autres cartes, en particulier sur la Pico.

```
10  void CC1100::reset(void)                // reset defined in cc1100 datasheet
11  {
12      digitalWrite(SS_PIN, LOW);
13      delayMicroseconds(10);
14      digitalWrite(SS_PIN, HIGH);
15      delayMicroseconds(40);
16
17      spi_write_strobe(SRES);
18      delay(1);
19  }
```

FIGURE 18 – Exemple de la fonction reset dans le driver issu de GitHub

Nous avions alors plusieurs options : recopier en les adaptant les fonctions de la librairie WiringPi* nécessaires pour le bon fonctionnement du driver, ou coder un driver inspiré de celui déjà codé pour Raspberry Pi mais en l'adaptant à Pico le tout en langage C/C++. La première option a été abandonnée après quelques essais car WiringPi s'appuie sur Linux or Pico n'a pas de noyau Linux. De plus, les fonctions de WiringPi étant très performantes, leur code nous était difficilement compréhensible. Imiter et reproduire sur la Pico les fonctions de WiringPi nous est donc apparu comme impossible.

La seconde option était bien plus abordable techniquement. Le problème majeur rencontré dans cette tâche a été le manque de temps. En effet, recoder dans son entièreté un driver pour un composant que nous ne connaissions finalement assez peu, pour une carte dont nous ne savions que peu de choses sur le fonctionnement aurait été extrêmement long. Un début de driver (Figure 19), inspiré des communications SPI donnée en exemple dans la documentation de Pico, a donc été commencé mais très vite abandonné, notamment en raison du manque d'informations sur les fonctions déjà à notre disposition sur la Pico.

Nous avons aussi tenté de coder un driver hybride dont une partie serait codée par nous mêmes en nous inspirant à la fois du driver du dépôt Github et des exemples de la documentation de Raspberry Pi Pico pour certaines fonctions et en réécrivant de manière simplifiée certaines fonctions de WiringPi pour d'autres. Cette méthode a échoué en raison de la trop grande complexité de WiringPi que nous n'avons pas réussi à reproduire dans nos fonctions.

```
static void cc1101_reset() {
    uint8_t buf[] = {SRES, 0x00};
    cs_select();
    spi_write_blocking(SPI_PORT, buf, 2);
    cs_deselect();
}
```

FIGURE 19 – Exemple de la fonction reset dans le driver codé en s'inspirant de la documentation de Raspberry Pi

De plus, étant donné que l'inspiration principale pour ce driver était le driver pour Raspberry Pi trouvé sur GitHub, nous avons eu de grosses difficultés à reproduire certaines fonctions de discussion SPI qui auraient pu nous être utiles. Malgré de nombreux essais avec différents GIT* et autres tutoriels d'installation, le résultat était toujours peu concluant. Jérémy Kalsron, une des personnes qui nous a beaucoup aidé pour ce projet, a eu l'idée de compiler avec l'IDE Arduino en utilisant une librairie spéciale. Nous nous sommes renseignés sur cette possibilité. Cette méthode étant belle et bien possible, nous avons trouvé des GIT proposant des librairies pour compiler comme cela. Un de ces GIT proposait une librairie de compilation Arduino pour Pico avec liaison SPI vers le récepteur qui nous convenait. Après des essais plus ou moins concluants, nous avons finalement réussi à compiler correctement et à faire fonctionner la liaison SPI. Plusieurs heures de travail plus tard, nous arrivons à écouter quelque chose qui s'apparente à une trame. La réception de cette trame marque la fin d'une grosse étape : la discussion SPI.

2.6.3 Librairie SmartRC

La librairie en question se nomme SmartRC*. Elle est conçue pour l'IDE Arduino afin que les composants comme CC1101 puissent communiquer par signaux radio. Deux modes sont proposés par SmartRC : le premier mode, appelé "minimal" permet une configuration rapide de l'écoute avec CC1101, ce mode permet de recevoir des signaux en bits provenant de la télécommande. Nous avons fait le choix d'utiliser un des modes qui permet d'exclure le câblage de GDO0 et GDO2 entre Pico et CC1101.

En sélectionnant 868.3 comme paramètre de la fonction setMHZ, 2 comme paramètre de la fonction setModulation et 4 comme paramètre de la fonction setSyncMode. (Figure 20) Le plus gros inconvénient de cette configuration est le fait que CC1101 ne filtre aucun signal écouté sur la fréquence 868.3Mhz, pour n'écouter et ne recevoir que les bits envoyés par une télécommande il faudra à l'avenir configurer le type de mot de synchronisation et de préambule attendu par CC1101. Le second mode s'appelle "advanced" il permet de configurer plus de paramètres et potentiellement d'écouter les télécommandes STEPHEAR.

Pour commencer nous savons que le choix de la modulation et de la fréquence est le même que celui de la fonction "minimal". Mais ce programme permet surtout de définir un mot de synchronisation avec la fonction setSyncWord ainsi que de décoder avec un code similaire à un code de Manchester, avec la fonction setManchester, ceci nous intéresse car la télécommande STEP-HEAR envoie un signal avec des symboles codés en transitions, exactement comme un code de Manchester.

```
ELECHOUSE_cc1101.Init();
ELECHOUSE_cc1101.setCCMode(1);
ELECHOUSE_cc1101.setModulation(2);
ELECHOUSE_cc1101.setMHZ(868.3);
ELECHOUSE_cc1101.setSyncMode(4);
ELECHOUSE_cc1101.setCrc(0);
ELECHOUSE_cc1101.setDRate(400);
```

FIGURE 20 – Initialisation du code Arduino

Ce programme permet aussi de choisir le data Rate en kbaud* pour l'écoute avec la fonction setDRate. Une fois l'écoute réussie avec le programme exemple de SmartRC il nous faut maintenant définir quels sont les paramètres permettant l'écoute et surtout la reconnaissance d'une télécommande.

L'étape suivante est de comprendre la trame reçue en SPI, la décoder pour comprendre la demande de la personne appuyant sur la télécommande.

2.7 Décodage de trame

L'IDE Arduino et la librairie trouvée pour lire ce que le récepteur reçoit nous permettent de voir ce que la CC1101 entend. Tout ceci est lisible sur la sortie série Arduino. Nous recevons une suite de chiffres compris entre 255 et 0 qui correspondent à une tension maximale et minimale. Donc si on dit que de 0 à 127 donne un 0 et que de 128 à 255 donne un 1, alors on obtient une suite de 1 et de 0 bien plus facile à traiter. Cependant on ne peut pas sortir directement le code hexadécimal voulu qui est 338A00 pour rappel. (Figure 21) Il faut trouver la bonne fréquence d'échantillonnage. En effet si l'on prend l'exemple du 3 : 0x3 = 0b0011 et ici on attend 1100, expliqué dans le paragraphe 2.2. Norme. Si l'on échantillonne trop large, on va recevoir plus de bits qu'il ne faut. Il faut donc réussir à en avoir assez pour savoir s' il n'y a qu'un ou plusieurs 1 derrière une suite de 1. Pour cela il faut faire deux choses : une recherche de la fréquence d'échantillonnage et un petit programme pour parser* la trame.

Revenons-en à la trame : la partie de code est codée sur 3 octets, chaque bit est codé en demi-bit comme le codage Manchester. Nous attendons donc 3x8x2=48 bits. Prendre seulement 48 bits serait source d'erreur. Il est préférable de prendre un multiple de 48, par exemple 5x48=144 bits. Un demi-bit de la tram attendue est de 250µs. On aimerait reçevoir 5 bits pour construire un demi-bit, soit 50µs par bit reçu. La librairie que nous utilisons prend en entrée un débit de données en kBaud donc de bit/s. On mettrait donc 1bits/50µs= 20 kbit/s = 20 kBaud.

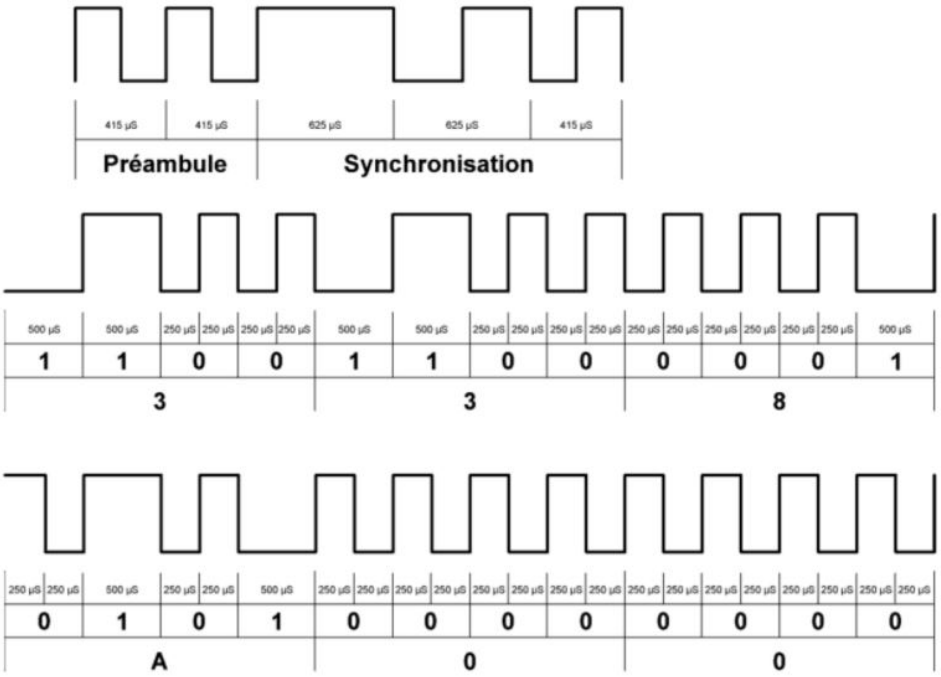

FIGURE 21 – Trame théorique d'après la Norme de communication [AFN15]

Le logiciel Universal Radio Hacker permet de décoder les trames écoutées à l'aide de la RTL-SDR. Pour cela il faut se rendre dans l'onglet "Analysis", depuis cet onglet on peut choisir le décodage qu'on applique au message reçu. Il faut auparavant s'assurer d'avoir bien réglé les paramètres d'écoutes (nombre d'échantillonnage, nombre d'échantillons par symbole, le type de modulation. De retour dans l'onglet Analysis, il faut ouvrir le menu déroulant "decoding" et choisir le code de ligne qui nous convient, dans notre cas il faut créer un nouveau code de ligne en cliquant sur les points de suspension. Il suffit ensuite de choisir Substitution et de remplir le tableau avec les enchaînements de bits que l'on veut traduire, dans notre cas 11 et 00 deviennent 1, tandis que 01 et 10 deviennent 0. Par exemple si on sélectionne la partie du signal qui correspond au mot de synchronisation et qu'on lui applique ce décodage, on obtient 1100 1100 0010 1010 0000 0000, ce qui en inversant bits de poids faible et bits de poids fort pour retrouver l'ordre initial donne 0011 0011 1000 1010 0000 0000 soit 338A00, le mot de synchronisation attendu.(Figure 21)

3 Pour aller plus loin

Il est important de parler un peu plus en détail de ce que nous voulions faire. Le temps imparti ne nous aura pas suffit à finaliser le projet. Cependant nous allons exposer la suite du travail à réaliser afin d'avoir une balise sonore opérationnelle.

3.1 La gestion du son

Afin que la balise soit capable d'avertir les utilisateurs d'un danger ou de la possibilité de traverser, il faut aussi que notre carte Pico ou Arduino soit capable de produire un son. Pour cela nous avons programmé une petite fonction permettant d'activer un buzzer piézoélectrique via GPIO. Bien entendu, le son d'une vraie balise n'est pas produit à l'aide d'un petit piezo-buzzer (Figure 22) tel que celui que nous avions à disposition. Toutefois, le piezo-buzzer peut nous permettre d'émettre plusieurs sons. Ce système de son rudimentaire peut nous permettre de présenter une version primitive de la balise sonore à la fin de notre projet. Un autre avantage du piézo-buzzer est son prix bien inférieur à celui d'un haut-parleur plus développé.

Un tel équipement bien qu'incapable de produire de son au-delà d'un certain seuil sonore répondrait aux attentes des utilisateurs ne voulant qu'une balise sonore épurée au prix le plus bas possible. De plus, malgré le son de piètre qualité émis par le buzzer, il est possible de jouer de petites mélodies simplistes (comme Joyeux Anniversaire) à l'aide de ce composant. L'une des possibles extensions de notre projet étant le développement de balises sonores pouvant être utilisées dans le cadre d'une exposition sonore, il faudra évidemment développer un autre système de production sonore pour pouvoir répondre à ce besoin.

FIGURE 22 – Exemple de buzzer piézoélectrique

3.2 Alimentation électrique

La balise que nous développons cherche à reproduire celle de STEP-HEAR, mais nous souhaitons y apporter des améliorations.L'une d'entre elles serait la possibilité d'installer une borne audio sans avoir à la brancher sur le secteur. Afin de répondre à ce besoin nous devons optimiser la consommation énergétique de notre borne. Jusqu'à présent nous n'avons pas trouvé de moyen de mettre la borne en veille lors d'une phase d'écoute passive.La principale difficulté face à ce problème est l'impossibilité d'écouter avec le CC1101 lorsqu'on met en veille la Pico.

Afin de rendre la borne audio portable, on doit aussi songer à installer une batterie dans le boîtier. L'une des possibilités pour permettre une autonomie plus longue, serait aussi de rajouter un dispositif de récupération d'énergie, par exemple une cellule photovoltaïque, sur la borne. Ainsi, on pourrait recharger la batterie sans avoir à brancher la borne sur le secteur. Néanmoins cette solution ne pourra pas être abordée dans le cadre de notre projet par manque de temps.

3.3 Télécommande

Pour pouvoir activer la balise, il faut une télécommande. Cependant, celles présentes sur le marché ne sont pas toujours accessibles au vu de leur complexité. (Figure 23) Pour rappel, la CC1101 est un récepteur mais aussi un transmetteur. On peut donc imaginer la création d'une télécommande. Une Raspberry Pi Pico, une CC1101, un buzzer et une batterie devraient suffir. (Figure 24) La grande difficulté de ce projet annexe serait l'écriture de la trame selon la norme. En effet, il faudrait correctement copier la norme, alors que pour rappel, cette dernière a différentes fréquences entre le préambule, le mot de synchronisation et les données.

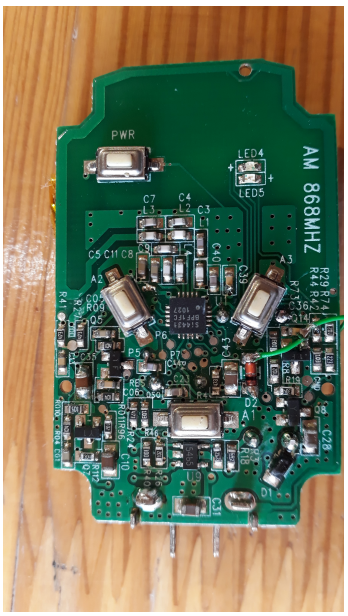

FIGURE 23 – Circuit intégré télécommande

FIGURE 24 – Composents intégrés à la télécommande

4 Conclusion

Notre objectif était de réaliser une balise sonore communiquant avec les télécommandes dont disposent généralement les malvoyants. Pour ce faire, nous devions utiliser des composants accessibles financièrement. En effet, ce projet a pour but d'être réalisable par tout le monde. Il faut donc un coût relativement bas et un tutoriel pour réaliser la balise soit même. De plus, nous avions comme contrainte la norme de communication entre la télécommande et la balise. Ce projet devait se finir par l'autosuffisance énergétique de la balise ainsi que des sons émis par un haut-parleur lors de l'utilisation de la télécommande.

Nous avons eu une proposition de microcontrôleur et de récepteur pour un coût réduit et une efficacité suffisante pour les tâches à accomplir. Nous avons donc commencé rapidement à nous occuper de la compilation du code qui permet la communication SPI avec le récepteur. Nous avons simultanément travaillé sur l'analyse de la norme grâce à un logiciel d'écoute radio et une télécommande.

Finalement, nous n'avons pas pu conclure ce projet. Nous avons finalisé la partie délicate du projet qui consiste à lire la trame grâce au récepteur et une sortie série. Cependant, la partie son n'a pas pu être réalisée, ainsi que la partie énergétique.

Glossaire

AFNOR : Association française de normalisation
Baud : Mesure du nombre de symboles transmis par seconde. Provient du patronyme d'Émile Baudot
GitHub, GIT : Logiciel de gestion de versions décentralisé
IDE : Integrated Development Environment
ISIMA : Institut Supérieur d'Informatique, de Modélisation et de leurs Applications
ISM : Industrielle, Scientifique et Médicale
LIMOS : Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes
Parser : Anglicisme. Parser signifie Analyseur. Ici utiliser pour parler du décodage de la trame
RX, TX : Mode de Réception et de Transmission
SDK : Software Development Kit
SDR : Software-Defined Radio
SmartRC : Libraire C++ permettant de communiquer en SPI avec la CC1101.
SPI : Serial Peripheral Interface
Thonny : IDE Python : Universal Serial Bus
WiringPi : Librairie Python déstinée aux microcontrôlleurs

Références bibliographiques

Références bibliographiques

[Tex13] TexasInstruments. "CC1101 Low-power Sub-1 GHz wireless transceiver". In : (2013).

[Tex14] TexasInstruments. "CC Debugger User's Guide". In : (2014).

[AFN15] AFNOR. "Dispositifs répétiteurs de feux de circulation à l'usage des personnes aveugles ou malvoyantes". In : (2015).

[Pi21] Raspberry Pi. "Raspberry Pi Pico C/C++ SDK". In : (2021).

[Ras21] RaspberryPi. "Raspberry Pi Pico Datasheet". In : (2021).