RAPPORT DE STAGE

Stagiaire au sein du LIUPPA

PONS Antonin

Avril 2024 à Juin 2024

Tuteur de stage : Angel ABENIA
Enseignant référent : Yannick LESPINE

Etablissement : IUT des Pays de l'Adour, Mont-de-Marsan – BUT 2 R&T (2023 – 2024)
Structure d'accueil : LIUPPA, Mont-de-Marsan

TABLE DES MATIÈRES

Remerciements **3**
Introduction **4**
Présentation de la structure **5**
Activités **7**
1. Déroulement des activités 7
2. Matériel mis à disposition 7
3. Rédaction de la bibliographie 9
4. Rédaction d'une procédure d'installation 11
5. Etude d'une télécommande 12
    a. Avec URH et RTL_433 12
    b. Avec Python 15
6. Etude d'une station météo 18
    a. Avec URH et RTL_433 18
    b. Avec Python 20
7. Bilan quantitatif 22

Conclusion **23**
Bilan Personnel **24**
Définitions **25**
Bibliographie **26**
Annexes **27**
Résumé (Abstract) **28**

Remerciements

Avant tout développement sur ce rapport de stage, je souhaiterais exprimer ma gratitude envers les personnes qui m'ont aidé et qui ont eu la gentillesse de me faire profiter de leurs savoirs durant ce stage.

Ainsi, je remercie Angel ABENIA (Professeur), mon tuteur de stage et Manuel MUNIER (Professeur), qui m'ont donné l'occasion de réaliser ma première expérience professionnelle au sein du LIUPPA et qui m'ont suivi et orienté avec beaucoup d'attention tout au long de mon stage.

Introduction

Actuellement en deuxième année de BUT Réseaux & Télécommunications, j’ai eu l’occasion d’effectuer un stage en entreprise au sein du LIUPPA dans le domaine de la radio 433 MHz.

Dans le cadre de mon rapport de stage, je vous présenterais mon étude approfondie sur les protocoles utilisés en 433 MHz en IoT.

L'objectif principal de ce stage était d'examiner les communications radio des appareils fonctionnant dans la bande de fréquence 433 MHz. La finalité de cette étude était non seulement de décoder les messages transmis, mais aussi d'enregistrer et de retransmettre des messages précédemment enregistrés. En outre, l'étude visait à encoder et à forger de nouveaux messages afin d'explorer les vulnérabilités potentielles des systèmes de communication (en réalisant par exemple du spoofing).

Pour atteindre ces objectifs, j'ai dû acquérir une compréhension approfondie de la technologie 433 MHz. Cela impliquait la rédaction de documentations techniques détaillées, la réalisation d'expériences pratiques et la conception de prototypes. Enfin, j'ai également rédigé des supports pédagogiques pour faciliter la compréhension de cette technologie par les étudiants.

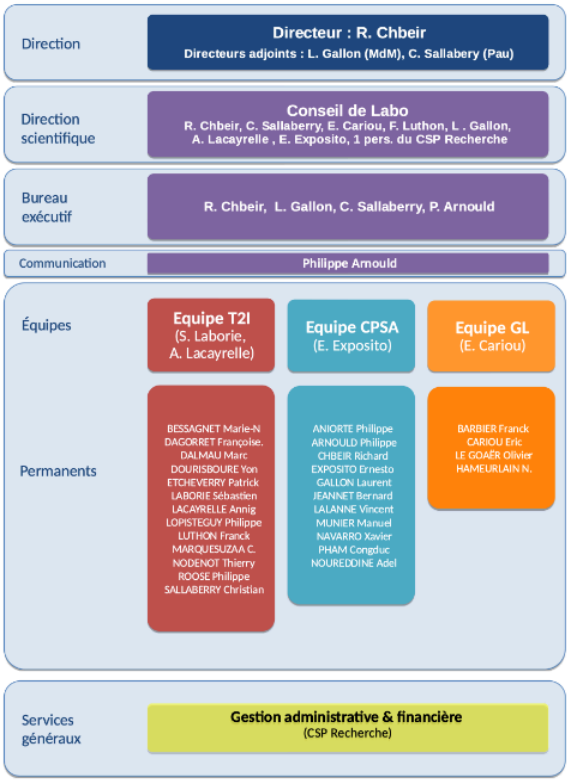
Présentation de la structure

Pour effectuer ce stage, j'ai été accueilli par le LIUPPA, c'est le laboratoire de recherche en informatique de l'Université de Pau et des Pays de l'Adour.

Le LIUPPA a été créé en 2000 et compte environ 50 membres et il est actuellement dirigé par Richard CHBEIR. Les membres du LIUPPA sont situés sur les trois campus de l'UPPA, que ce soit à Bayonne-Anglet, à Pau ou à Mont-de-Marsan.

Le LIUPPA mène des recherches appliquées dans le domaine de l'informatique. Parmi ses principaux domaines d'étude, on peut retrouver le génie logiciel, la sécurité informatique, les systèmes d'informations, les réseaux et protocoles.

Voici l'organigramme du LIUPPA (2020):



Organigramme du LIUPPA

Comme on peut le voir sur l'organigramme, Le LIUPPA est structuré en 3 équipes :

L'équipe Architecture des Systèmes Cyber-Physiques (ASCP) se concentre sur la gestion des données, la conception de systèmes et la coordination des équipements, en mettant l'accent sur la sécurité et la confidentialité. L'équipe adopte une approche globale, en utilisant des compétences dans des domaines tels que l'ingénierie des systèmes, l'intégration et l'interopérabilité sémantique, et la gestion autonomique des systèmes coopératifs.

Le Traitement des Informations pour l’Adaptation de l’Interaction au Contexte et à l’Utilisateur (T2I) se focalise sur les éléments externes et contextuels d'un système cyber-physique. L'objectif est de concevoir, implémenter et déployer des applications interactives et adaptatives qui traitent des données diverses. Cela permet de valoriser l'information et de faciliter les interactions de l'utilisateur en lui fournissant les informations les plus pertinentes.

Le Génie Logiciel (GL) s'intéresse aux langages de spécification et de modélisation semi-formelle pour la conception de logiciels de qualité. Avec l'augmentation du nombre de logiciels et leurs nouvelles caractéristiques (cloud, big data, cyber-physique...), de nouvelles approches de conception sont nécessaires, ce qui est abordé par cette équipe.

Activités

1. Déroulement des activités

Durant ce stage, j'ai eu l'occasion d'effectuer un certain nombre d'activités afin de produire des documents techniques et pédagogiques sur la radio SDR (Software Defined Radio) 433 MHz.

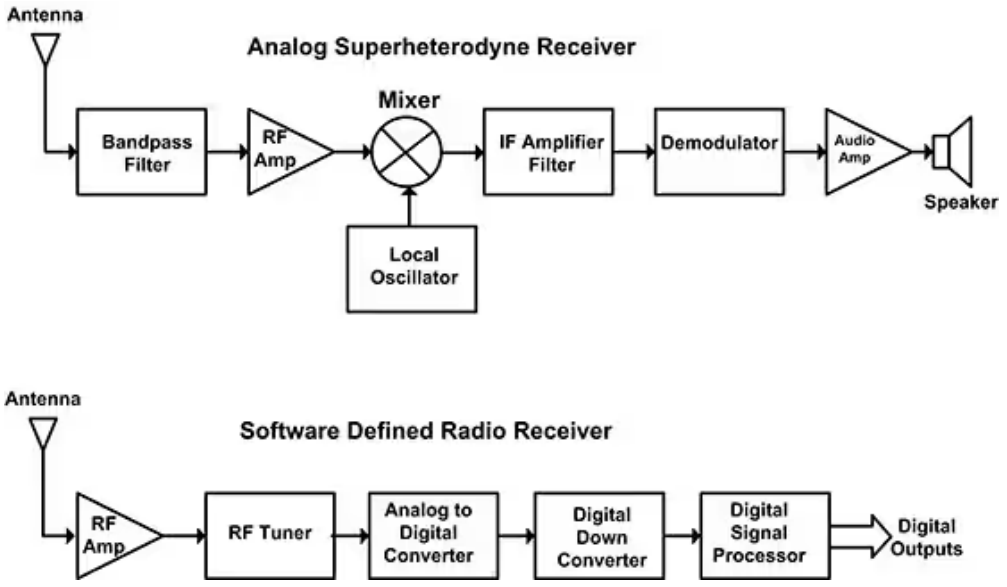
Tout d'abord, j'ai pu rédiger une bibliographie des logiciels SDR à essayer. Puis, j'ai eu l'occasion de tester ces logiciels et de produire un guide d'installation sur les machines virtuelles de l'IUT. Ensuite, je me suis penché sur le protocole de communication utilisé par des télécommandes et j'ai commencé à rédiger des programmes Python permettant d'automatiser la réception et l'émission de trames altérées. Enfin, j'ai réutilisé les scripts que j'avais écrits précédemment pour les faire fonctionner avec une station météo.

2. Matériel mis à disposition

Durant ce stage, j'ai eu à utiliser un émetteur-récepteur SDR afin de recevoir et d'émettre des signaux hertziens.

La radio SDR consiste, contrairement à la radio analogique, à numériser le plus tôt possible le signal afin de réduire le nombre de composants électroniques utilisés. En effet, la radio SDR consiste à amplifier le signal puis à le transposer autour de 0 Hz, et enfin de le numériser afin de laisser aux logiciels le soin de démoduler le signal. Le principal avantage de la radio SDR est que les logiciels utilisés peuvent être changés facilement dans la chaîne de transmission, ainsi le même matériel peut être utilisé pour un grand nombre de communications radios différentes.

Voici une illustration du fonctionnement d'un récepteur SDR face à un récepteur analogique:



[Récepteur analogique vs SDR (Digi-Key Electronics)]

Comme on peut le voir, le récepteur SDR (figure du bas) convertit directement en numérique (avec Analog to Digital Converter) le signal détransposé (avec RF Tuner).

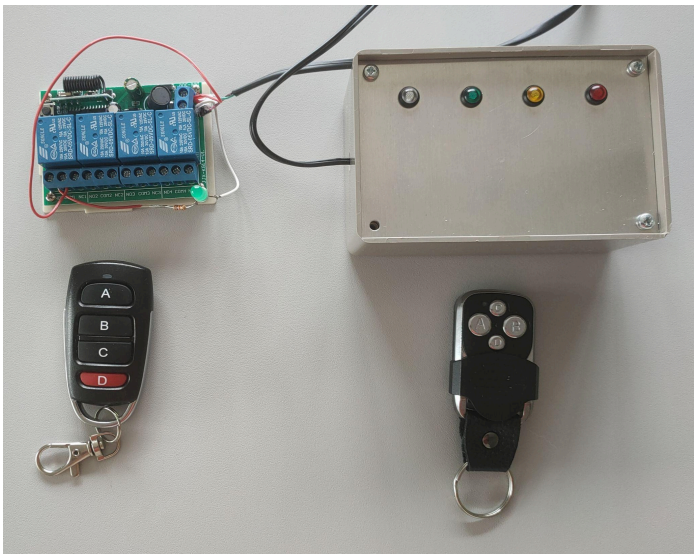
Matériel SDR mis à disposition:

ADALM-PLUTO, récepteur/émetteur SDR

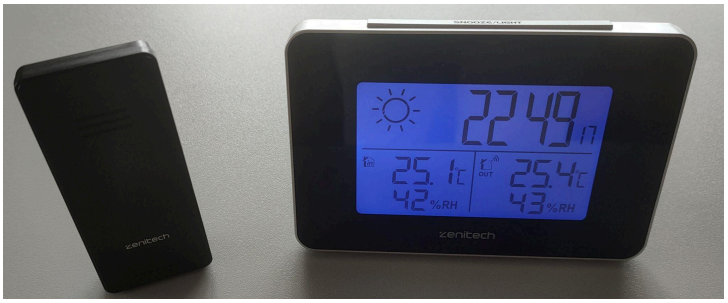
Clé RTL-SDR Nooelec

Dans notre cas de figure, nous avons choisi d'utiliser l'ADALM-PLUTO car celui-ci permet contrairement au Nooelec d'émettre. De plus, celui-ci est moins sensible au bruit.

Concernant le matériel à «hacker», on peut retrouver des télécommandes avec des récepteurs. Pour le récepteur de droite, une led différente s'allume selon le bouton appuyé sur la télécommande.



On peut aussi retrouver une station météo, les informations sont envoyées par le boîtier à gauche de l'image:



3. Rédaction de la bibliographie

La première étape de mon stage fut de rédiger une bibliographie regroupant les différents logiciels SDR que l'on peut rencontrer et qu'il serait intéressant d'utiliser.

Pour choisir les logiciels à essayer, j'ai dû me soumettre à certaines contraintes. Tout d'abord, les logiciels choisis devaient être compatibles avec Linux et Windows, de plus ceux-ci devaient être gratuits et de préférence open source afin d'éviter des frais de licence.

Parmi tous les logiciels existants, certains se prêtent mieux que d'autres à l'étude et la transmission de signaux d'IoT en 433 MHz.

On peut retrouver RTL_433, ce logiciel permet de capturer les échanges IoT sur la bande 433 MHz et de les analyser afin de trouver leur protocole. Ce logiciel supporte des dizaines de protocoles, cependant celui-ci ne détecte pas correctement la télécommande que nous utilisons, voici ce qu'il retourne:

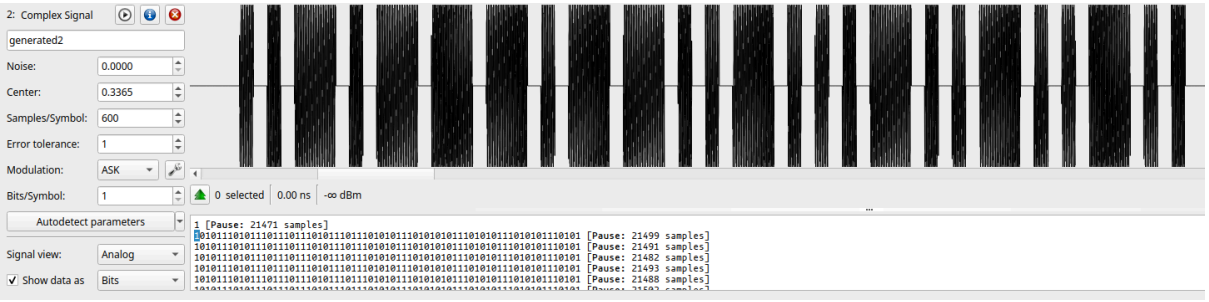
```
supervisor@ubuntu2204:~/Téléchargements/rtl_433-23.11$ rtl_433 -d "driver=plutosdr,uri=ip:192.168.2.1"
rtl_433 version 23.11 (2023-11-28) inputs file rtl_tcp RTL-SDR SoapySDR
 SoapySDR  Unable to scan local: -19

Using device ADALM-PLUTO: ad9361-phy,model=ad9363a ad9361-phy,xo_correction=40000051 backend_version=0.25 (
68.2.1 library_version=0.25 (git tag: b6028fd) local,kernel=5.15.0-175882-ge14e351533f9 uri=ip:192.168.2.1
Found 1 antenna(s): A_BALANCED
Found 1 gain(s): PGA 0 - 73 (step 0)
Found 1 frequencies: RF
Found 1 frequency range(s): 70000000 - 6000000000 (step 0)
Found 1 sample rate range(s): 260417 - 61440000 (step 0)
Found 11 bandwidth range(s): 200000 - 200000 (step 0) 1000000 - 1000000 (step 0) 2000000 - 2000000 (step 0)
00 - 9000000 (step 0) 10000000 - 10000000 (step 0)
Found current bandwidth 2100000
Found 4 stream format(s): CS8 CS12 CS16 CF32
Found native stream format: CS16 (full scale: 2048.0)
 SDR  Tuner set to automatic gain.
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
time      : 2024-05-02 11:40:58
model     : Smoke-GS558  id        : 18587
unit      : 20           learn     : 0            Raw Code  : 891374
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
time      : 2024-05-02 11:40:59
model     : Smoke-GS558  id        : 18587
unit      : 20           learn     : 0            Raw Code  : 491374
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
time      : 2024-05-02 11:41:00
model     : Smoke-GS558  id        : 18587
unit      : 20           learn     : 0            Raw Code  : 291374
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
time      : 2024-05-02 11:41:01
model     : Smoke-GS558  id        : 18587
unit      : 20           learn     : 0            Raw Code  : 191374
^CSignal caught, exiting!
supervisor@ubuntu2204:~/Téléchargements/rtl_433-23.11$
```

Ainsi, si on souhaite étudier le protocole utilisé par la télécommande, il va falloir utiliser des logiciels permettant d'analyser des couches plus basses de la chaîne de transmission.

Pour ce faire, on peut utiliser un outil comme URH. Ce logiciel permet de démoduler et de décoder des signaux afin de retrouver les données binaires.

Voici à quoi ressemblent les données envoyées par la télécommande que l'on reçoit:



Sur la partie haute de la fenêtre, on peut voir le signal analogique, en cliquant sur le bouton analog en bas à gauche, on peut afficher sa version démodulée. Sur la partie basse, on peut lire les données binaires décodées.

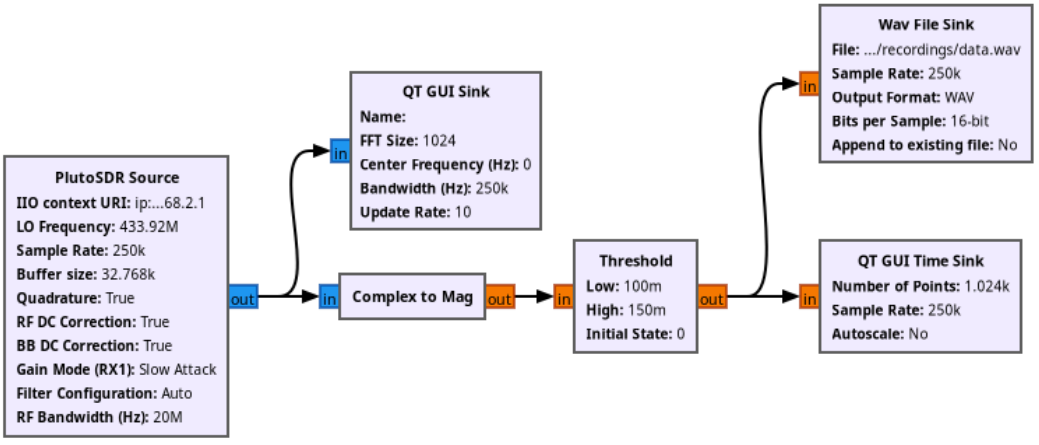
De plus, ce logiciel possède aussi un onglet «analyse» permettant de décoder le signal en bande de base de différentes manières. Par défaut, on peut décoder le signal en NRZ, Manchester, ou Manchester différentiel, mais il est possible de créer notre propre décodeur.

URH possède aussi un onglet «générateur» permettant d'envoyer une suite de données binaires modulée en ASK, FSK ou PSK et codées selon de décodeur utilisé dans l'onglet «analyse».

Mais, le logiciel URH n'est pas efficace pour automatiser la réception, il va alors falloir utiliser un outil plus complet et complexe tel que GNURadio.

GNURadio est un logiciel libre et open source de traitement du signal pour les systèmes de communication sans fil. C'est un environnement de développement basé sur une interface graphique qui permet de concevoir et de tester des systèmes de communication sans fils en utilisant des blocs fonctionnels prédéfinis ou en créant ses propres blocs en Python.

Exemple avec un diagramme de flux permettant d'enregistrer les données reçues dans un fichier wav:



Dans ce diagramme, on récupère les données reçues à l'aide du bloc PlutoSDR Source (signal reçu par l'ADALM-PLUTO). Puis, on convertit les nombres complexes en sortie du bloc source en nombre réel afin de pouvoir les enregistrer par la suite. Le bloc Threshold permet de définir les paliers, si le niveau du signal est en dessous de la valeur Low, le bloc Threshold renverra 0, sinon si le niveau est au-dessus de High, le bloc renverra 1. Ainsi, en sortie on aura un fichier wav contenant le signal reçu en bande de base.

4. Rédaction d'une procédure d'installation

Après avoir fait cette bibliographie, il m'a été demandé d'écrire un support d'installation des logiciels SDR pour Windows 10 et Ubuntu 22.04. La procédure mise en place est trouvable dans le document d'installation présent sur le dépôt des livrables que j'ai fournis durant mon stage.

Pour Ubuntu, la procédure consiste à installer avec le gestionnaire de paquets APT SoapySDR et ses dépendances, libiio, gnuradio et RTL_433. Puis, il a aussi fallu compiler depuis la source le module PlutoSDR pour SoapySDR.

Libiio est une librairie utilisée pour interfacer avec l'ADALM-PLUTO. Elle va notamment fournir les blocs SoapySDR Source et SoapySDR Sink pour la réception et l'émission via GNURadio.

SoapySDR est une API permettant d'interagir de façon homogène avec n'importe quel matériel SDR.

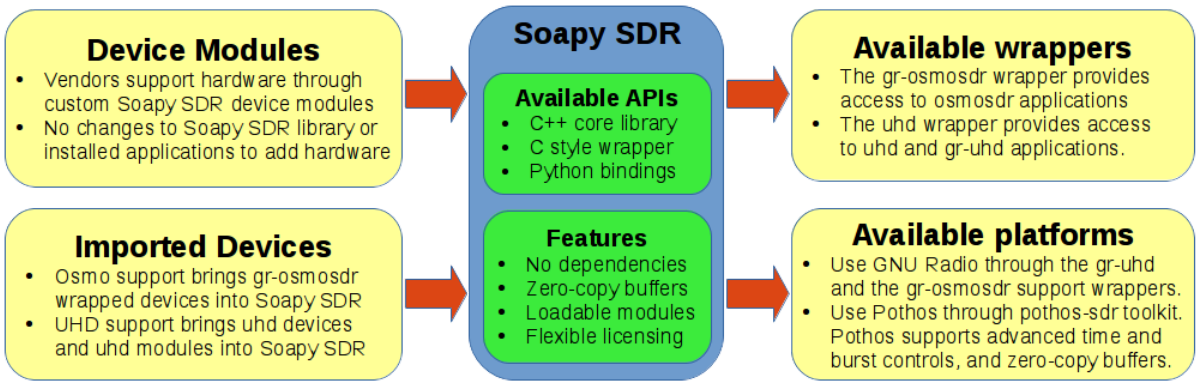


Illustration du fonctionnement de SoapySDR

Comme on peut le voir ci-dessus, Soapy SDR se base sur les modules des appareils IoT (à gauche), comme PlutoSDR pour SoapySDR, pour fournir un accès homogène à différents logiciels (à droite), comme GNU Radio.

Enfin, pour l'installation de URH, je suis passé par pipx pour l'installer.

Pour Windows 10, il suffit d'installer PothosSDR et le driver de l'ADALM-PLUTO. En effet, l'environnement PothosSDR inclut GNURadio.

Puis, pour URH, il faut l'installer à partir de l'exécutable disponible sur github.

Enfin, pour RTL_433, la procédure fut plus complexe à trouver, en effet, il faut télécharger l'archive RTL_433 depuis github puis déplacer le fichier rtl_433-rtlsdr-soapysdr.exe vers le répertoire C:\Program Files\PothosSDR\bin et exécuter RTL_433 depuis cet emplacement.

A la base, rtl_433 ne fonctionnait pas en l'exécutant depuis l'archive téléchargée, j'ai trouvé cette solution ici.

5. Etude d'une télécommande

a. Avec URH et RTL_433

Comme nous l'avons vu précédemment, RTL_433 ne détecte pas la télécommande correctement. Cependant, en changeant de matériel SDR (une clé RTL-SDR), je me suis rendu compte que RTL_433 pouvait interpréter convenablement les données envoyées par la télécommande.

Voici ce que RTL_433 renvoie:

```
time      : 2024-05-28 14:30:31
model     : Akhan-100F14 ID (20bit): 0x2ec89
Data (4bit): 0x1 (Lock)
```

Ces données paraissent beaucoup plus cohérentes. En effet, lorsque l'on appuie sur un bouton différent de la télécommande, le champ Data varie et prend les valeurs 1, 2, 4 ou 8. On observe alors que le protocole décodé est le protocole «Akhan-100F14».

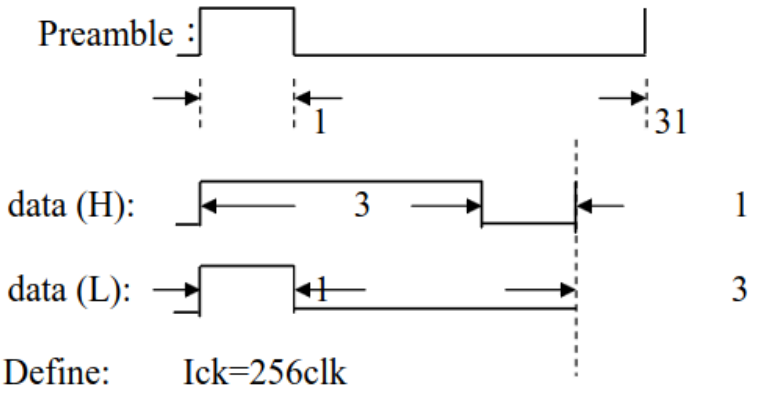
En explorant le code source de RTL_433, j'ai pu retrouver ces informations sur le protocole:

HS1527 OTP Encoder

HS1527 Output format: (Compatible:EV1527,RT1527,FP1527)

Output data frame:

Preambl e	C0 ~ C19 (1 million codes)	D0	D1	D2	D3

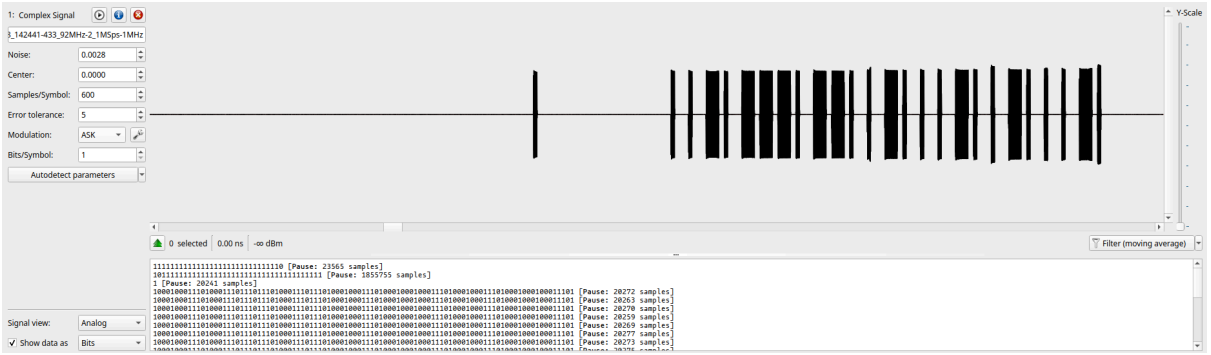


Extrait de la documentation technique du protocole de la télécommande

Désormais, on a une idée très claire du format de la trame et du type de codage utilisé. Chaque trame commence par un préambule composé d'un front haut suivi par 31 fronts bas. Un 1 est codé par trois fronts hauts suivis d'un front bas et un 0 est codé par un front haut suivi de trois fronts bas. Ce type de codage s'appelle la MLI (Modulation en Largeur d'Impulsion).

Pour les données, les 20 premiers bits de la trames codent l'identifiant et les quatres bits suivants codent l'action à effectuer.
Ensuite, j'ai dû utiliser URH afin de déterminer la durée d'un bit.

Voici la capture des données de la télécommande:

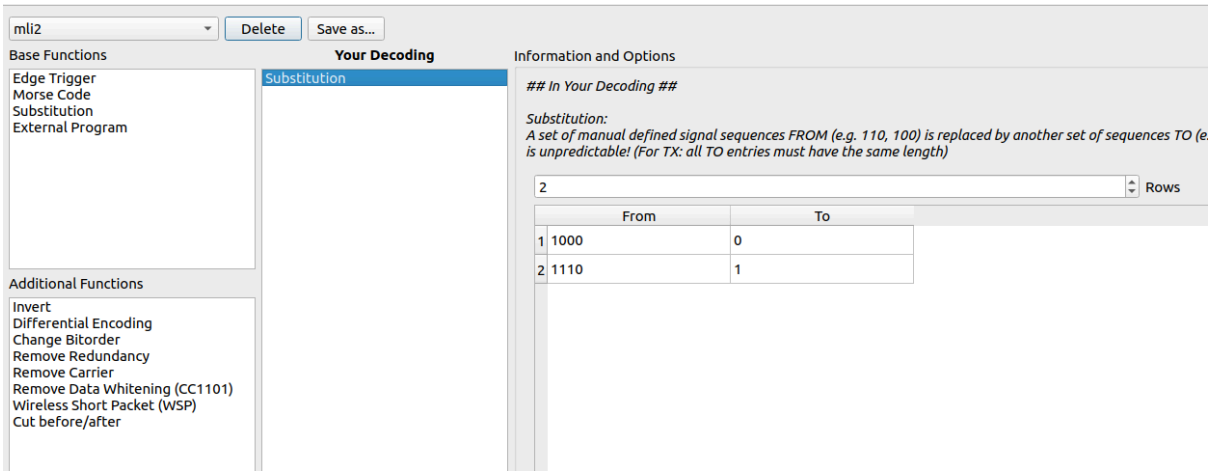


On observe la présence d'un préambule, et on peut aussi se rendre compte que les données trouvées dans la documentation du protocole ont l'air exactes. Sur l'écran de gauche, on peut retrouver un champ «Samples/Symbol», cela signifie échantillons par symboles. On observe que lorsque cette valeur est définie à 600, un zéro binaire est décodé comme étant la suite de symboles «1000» (un front haut pour trois fronts bas) et un un binaire comme étant «1110» (trois fronts haut pour un front bas). Ainsi, si on veut étudier les données binaires réelles, il faut multiplier le nombre d'échantillons par symbole par 4. Alors, on peut décoder les données suivantes: 2ec891 (en hexa). Ces données sont cohérentes avec celles de RTL_433. En effet, les quatres premiers symboles des données en hexadécimal représentent l'identifiant de la télécommande, et le dernier contient l'action à exécuter.

Pour interpréter la valeur d'un bit, URH se base sur la valeur moyenne. Ainsi, si le signal est haut seulement un quart du temps (inférieur à la moitié du temps), URH va interpréter un zéro, alors que s'il est haut trois quarts du temps (supérieur à la moitié du temps), URH va interpréter un un.

Cependant, on peut aussi décoder les données en utilisant l'onglet «analyse» et en utilisant un décodeur particulier. Comme dit précédemment, il n'existe pas de décodeur MLI par défaut, il va donc falloir le créer.

Pour ce faire on peut, par exemple utiliser le bloc substitution, ce qui donne ceci:



Comme on peut le voir, un symbole positif suivit de trois symboles nuls donneront un zéro, alors que trois symboles positifs suivis d'un symbole nul donneront un un.

Voici à quoi ressemblent nos données avant décodage:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
1	1																																														
2	1	1	1	0	1	1	1	0	1	0	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1
3	1	0	0	0	1	0	0	0	1	1	1	0	1																																		
4	1	1	1	0	1	1	1	0	1	0	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1
5	1	1	1	0	1	1	1	0	1	0	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1
6	1	1	1	0	1	1	1	0	1	0	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1

Après décodage:

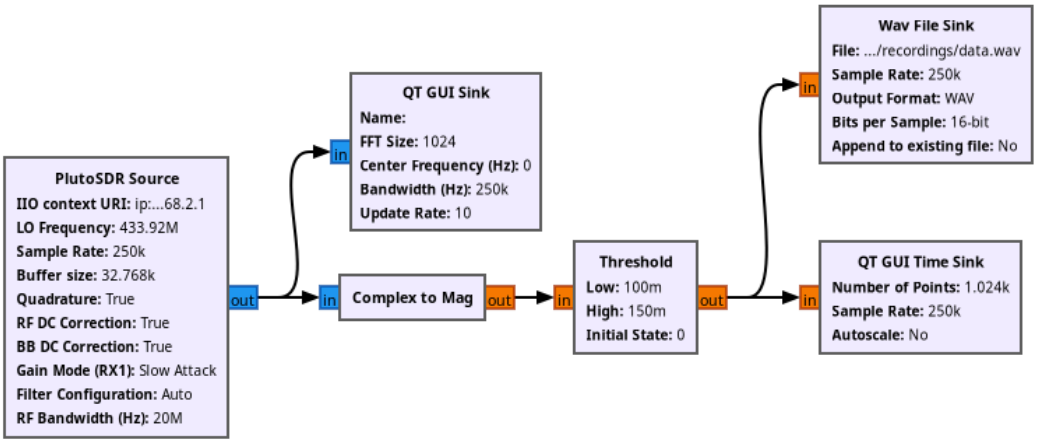
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	1																								
2	1	1	0	1	0	1	1	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	1	0
3	0	0	1	0																					
4	1	1	0	1	0	1	1	0	1	1	1	1	0	1	0	1	0	1	0	0	0	0	1	1	
5	1	1	0	1	0	1	1	0	1	1	1	1	0	1	0	1	0	1	0	0	0	1	1		
6	1	1	0	1	0	1	1	0	1	1	1	1	0	1	0	1	0	0	0						

Ainsi, à l'aide de ce décodeur, on va aussi pouvoir faire l'opération inverse lors de l'envoi en encodant les données binaires en MLI.

b. Avec Python

Pour automatiser les tâches de réception et d'émission, j'ai eu besoin d'utiliser GNURadio. En effet, URH possède un outil en ligne de commande efficace pour automatiser certaines tâches, cependant celui-ci est difficile à prendre en main car peu documenté.

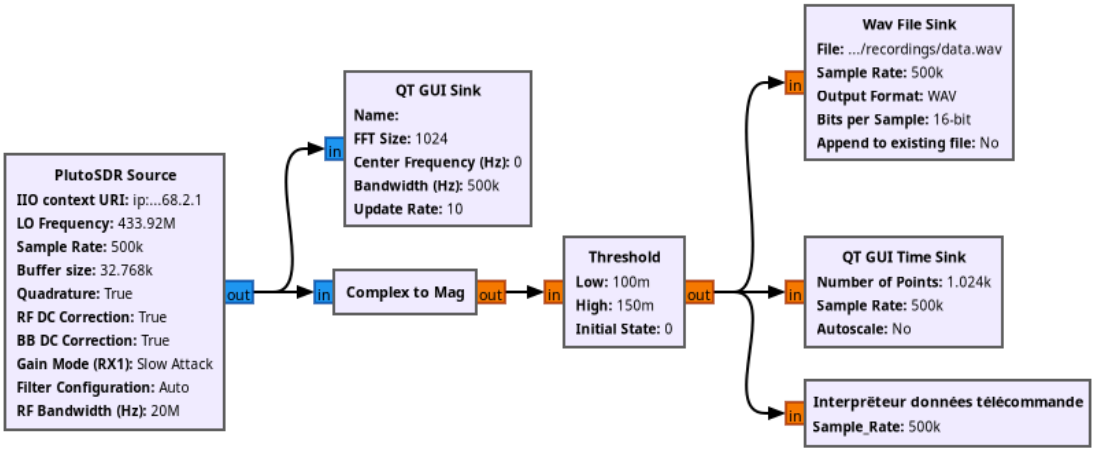
J'ai commencé par créer un flowgraph sur GNURadio permettant de récupérer le signal carré de la transmission.



Ce diagramme a déjà été expliqué dans la partie Bibliographie, il permet de stocker le signal carré dans un fichier wav.

A l'aide du diagramme ci-dessus, j'ai pu écrire un premier programme Python séparé de GNURadio afin d'interpréter les données reçues dans un fichier WAV. Cependant, le problème de cette méthode est que le programme Python ne décode pas les données en temps réel. Ainsi, pour ce faire, il a fallu créer un bloc Python sur GNURadio afin d'envoyer les échantillons reçus directement au programme Python plutôt que de passer par un fichier WAV.

Voici le diagramme contenant le code Python pour la réception:



Le bloc «Interpréteur données télécommande» est un bloc écrit en python par mes soins.

Voici un extrait du bloc exécuté par GNURadio pour la réception:

```
    def work (self, input_items, *args, **kwargs):
        samples = input_items [0]
        for i in samples:
            # Si l'échantillon parcourut est un 1 et qu'il y a eu plus
de trois zéros
            if i and self.zero_bits > 3:
                # Si la durée du dernier blanc correspond environ à
celle d'un préambule
                if self.zero_bits > self.sps*28 and self.zero_bits <
self.sps*34:
                    self.decode()
                    self.trame = ""
                # Sinon si elle correspond environ à celle d'un 0
                elif self.zero_bits > self.sps*2:
                    self.trame += "0"
                # Sinon si elle correspond environ à celle d'un 1
                elif self.zero_bits < self.sps*2:
                    self.trame += "1"
                self.zero_bits = 0
            elif not i:
                self.zero_bits += 1
        return len(samples)
```

Code complet du bloc

Ce programme parcourt les échantillons que GNURadio lui envoie. Il compte le nombre d'échantillons nuls jusqu'à tomber sur un échantillon non-nul. Puis, selon le nombre d'échantillons nul, il détermine si la valeur actuelle est un 1, un 0 ou un préambule. Si c'est un préambule, il affiche la trame en construction, sinon, il ajoute un 1 ou un 0 à la fin de la trame.

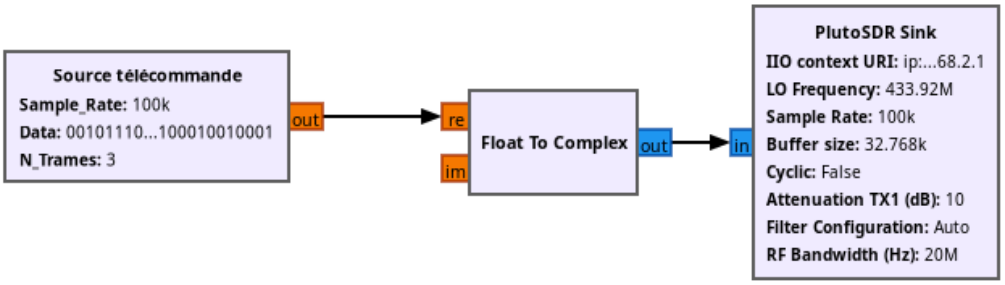
Voici l'affichage retourné par le programme de réception:

```
id: 2ec89; action: 1
id: 2ec89; action: 2
id: 2ec89; action: 4
id: 2ec89; action: 8
```

Pour le programme d'émission, j'ai d'abord réalisé une version utilisant urh_cli, puis une autre basée sur GNURadio.

La version utilisant urh_cli récupère les paramètres fournis par l'utilisateur en ligne de commande et génère une commande urh_cli puis l'exécute. Ce programme fournit une interface simple pour utiliser urh_cli comme générateur de trames commande.
Cependant, j'ai aussi fait une version GNURadio car l'avantage de ce logiciel est qu'on le retrouve sur un plus grand nombre de systèmes que URH. De plus, les performances de GNURadio sont bien meilleures lors du début de l'émission.

Voici le flowgraph GNURadio:



Voici un extrait du code du bloc exécuté par GNURadio:

```
    def work(self, input_items, output_items):
        # out est le tableau des valeurs à envoyer en sortie du bloc
        out = output_items[0]
        length = len(out)
        i = 0
        # Tableau temporaire qui écrasera le tableau out une fois
construit
        out_arr = []
        # Si toutes les données ont été transmises
        if self.last >= len(self.arr) and not self.emitting:
            os.system(f"kill {os.getpid()}")
        elif self.last >= len(self.arr) and self.emitting:
            self.emitting = False
            self.last = 0
            self.arr = np.array([0 for _ in range(32768)], dtype="f")

        # Tant le le tableau des valeurs de sortie n'est pas plein
        while i < length:
            out_arr.append(self.arr[(self.last+i)%len(self.arr)])
            i += 1
        self.last += i
        # On écrit sur le tableau des valeurs de sortie
        out[:] = np.array(out_arr, dtype=np.float32)
        return length
```

Code complet du bloc

Ce programme parcourt la liste self.arr et ajoute chaque valeur au tableau de sortie. Le tableau de sortie correspond au buffer utilisé par GNURadio. Cependant, le PlutoSDR utilise une taille de buffer différente (32768), ainsi après avoir traité toutes les données présentes dans le tableau self.arr, il faut envoyer des échantillons nuls afin de remplir le buffer du PlutoSDR.

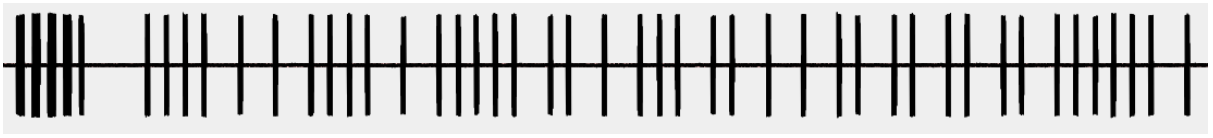
Maintenant que j'ai pu «usurper» une télécommande en émettant des trames générées par un programme et en recevant les trames envoyées par les autres télécommandes, j'ai pu appliquer le même raisonnement pour pirater une station météo.

6. Etude d'une station météo

a. Avec URH et RTL_433

Afin d'étudier le protocole utilisé par la station météo qui m'a été fourni, j'ai décidé de commencer par analyser le signal envoyé par la sonde météo en bande de base afin de déterminer le type de codage utilisé.

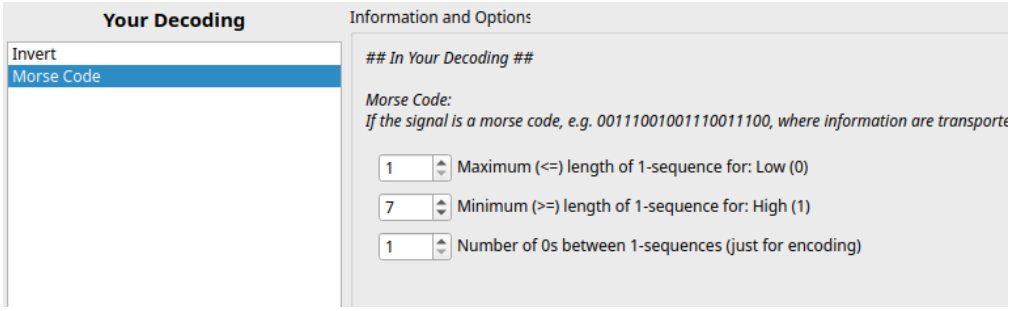
En utilisant URH, on peut observer ces données en réception:



On se rend compte que cette trame est composée d'un préambule, qui est répété au début de chaque trame, puis d'une suite d'informations. A l'aide de cette capture, on peut en déduire que la valeur d'un bit dépend de la longueur du blanc après une impulsion. En définissant le nombre d'échantillon par symboles au nombre d'échantillon sur une impulsion, on se rend compte que les données sont encodées par une impulsion d'une longueur l suivie d'un blanc d'un longueur 3l ou 7l.

Ainsi, pour décoder les données sur URH, un décodeur par substitution ne fonctionnera pas. En effet, vu que la longueur des fronts montants est la même pour un 0 ou un 1, on pourrait décoder des blancs courts à l'intérieur des blancs longs. Cependant, on peut utiliser le bloc d'inversion et le bloc morse afin de décoder ces données.

Voici à quoi ressemble le décodeur pour ce type de codage:



Voici les données décodées avec le nouveau décodeur:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1																																									
2	0	0	0	0																																				
3	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	0	1	1	0	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1
4	0	0	0	0																																				
5	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	0	1	1	0	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1
6	0	0	0	0																																				
7	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	0	1	1	0	0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1
8	0	0	0	0																																				

Maintenant que nous avons pu déterminer comment étaient codées les données, nous allons pouvoir identifier les informations que nous rencontrons ici. Pour ce faire, j'ai décidé d'utiliser RTL_433 afin de détecter le protocole utilisé par la sonde et d'extraire les données du signal.

Voici que que l'on obtient en sortie de RTL_433:

```
time      : 2024-05-22 11:03:44
model     : inFactory-TH ID        : 28
Channel   : 1            Battery   : 1            Temperature: 72.90 F      Humidity  : 54 %
Integrity : CRC
```

On observe tout d'abord dans le champ model que le nom du protocole est inFactory-TH.

En explorant le code source de RTL_433, j'ai pu trouver des informations sur le protocole utilisé:

```
Transmissions includes an id. Every 60 seconds the sensor transmits 6 packets:

    0000 1111 | 0011 0000 | 0101 1100 | 1110 0111 | 0110 0001
    iiii iiii | cccc ub?? | tttt tttt | tttt hhhh | hhhh ??nn

- i: identification // changes on battery switch
- c: CRC-4 // CCITT checksum, see below for computation specifics
- u: unknown // (sometimes set at power-on, but not always)
- b: battery low // flag to indicate low battery voltage
- h: Humidity // BCD-encoded, each nibble is one digit, 'A0' means 100%rH
- t: Temperature // in °F as binary number with one decimal place + 90 °F offset
- n: Channel // Channel number 1 - 3
```

Commentaire du module infactory.c

Tout d'abord, les huit premiers bits de la trame sont les bits d'identification. Dans notre capture RTL_433 l'identifiant est de 28, et dans URH, les huit premiers bits de données correspondent à 28 dans le cas où les zéros sont codés par des blancs courts et les uns par des blancs longs.

En lisant la documentation ci-dessus, on se rend compte que l'identifiant est suivit de 4 bits de CRC, d'un bit sur l'état de la batterie, de quatre bits inconnus, puis de la température exprimée en F°*10+90 (elle est codée comme ceci afin de coder les valeurs négatives et un chiffre après la virgule en utilisant du binaire naturel), et de l'humidité codée en BCD (Binaire Codé Décimal), puis enfin du numéro du canal utilisé.

En reprenant les données binaires décodées depuis URH, on peut déduire de la trame les informations suivantes manuellement:

```
0001 1100 | 0110 0001 | 0110 0101 | 1101 0101 | 0100 0001
iiii iiii | cccc ub?? | tttt tttt | tttt hhhh | hhhh ??nn
id:       4+8+16=28
CRC:      0110
battery:  0 (not low)
hum:      0101 0100 (BCD 4 bits) => 54%
temp:     0110 0101 1101 = 1629 => 1629/10-90 = 72.9°F
chan:     01
```

b. Avec Python

Les scripts Python rédigés pour la station météo sont semblables à ceux de la télécommande. Cependant, il faut changer le nombre d'échantillons par symboles, le type de codage (Modulation par largeur de blanc), et la façon dont les données sont décodées.

La difficulté principale pour l'adaptation des programmes de réception ou d'émission fut le calcul du CRC. Pour ce faire, je me suis inspiré du code C du décodeur du protocole inFactory de RTL_433. Dans ce code, on observe l'utilisation de la fonction crc4. En la recherchant dans le dépôt de RTL_433, j'ai pu retrouver le code source de cette fonction. Alors, en traduisant ces bouts de code en python, j'ai pu implémenter le calcul et la vérification du CRC dans mes programmes.

```python
def crc4(message: list, n_bytes: int, polynomial: int, init: int)->int:
    remainder = init << 4
    poly = polynomial << 4
    for i in range(n_bytes):
        remainder ^= message[i]
        for bit in range(8):
            if remainder & 0x80:
                remainder = (remainder << 1) ^ poly
            else:
                remainder = (remainder << 1)

    return remainder >> 4 & 0x0f
```

On peut retrouver ci-dessus le programme de calcul du CRC. Cette fonction prend en paramètres un tableau d'octets message, le nombre d'octets de la trame, le polynôme à utiliser et la valeur initiale pour le calcul du CRC.

Voici le résultat renvoyé par le programme de réception:

```
trame      : 11010011001000010110100000100011100100001
id         : d3
temperature: 24.8°C
humidity   : 39%
battery    : 0
channel    : 1
```

J'ai aussi ajouté une fonctionnalité permettant d'afficher les données reçues dans la sortie standard en JSON, ainsi qu'une autre permettant de lire les données à envoyer depuis l'entrée standard afin de pouvoir modifier les données reçues à la volée et de les envoyer par la même occasion.

Ainsi, ces programmes connectés en utilisant le pipe (fonction shell permettant de rediriger la sortie standard d'un programme vers l'entrée standard d'un autre programme) permettent par exemple d'ajouter un décalage de température ou d'humidité aux données reçues, puis de les émettre de manière à ce que le récepteur affiche les données altérées.

Tout d'abord, on peut lancer la commande permettant d'enregistrer les trames modifiées dans un fichier:

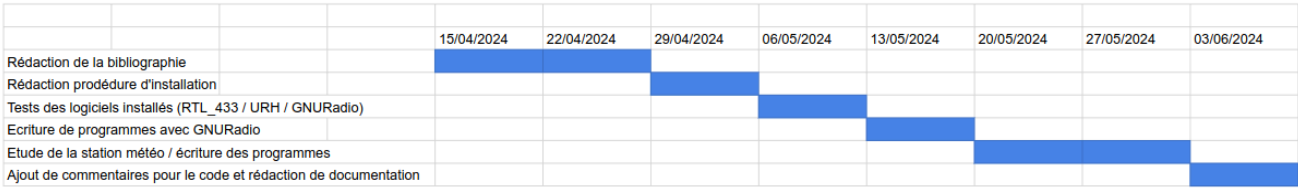
```
./reception/cli.py --threshold 0.4 0.5 --json | ./transceive/cli.py > file.txt
```

Le programme `transceive/cli.py` permet de modifier les données JSON envoyées par le programme de réception et d'envoyer ces nouvelles données dans l'entrée standard.

Puis, émettre les données enregistrée dans ce fichier:

```
tail -f file.txt | ./emission/cli.py --stdin
```

7. Bilan quantitatif



Durant les deux premières semaines de mon stage, j'ai pu rédiger la bibliographie en autonomie à distance de mon lieu de travail afin de découvrir les fondamentaux de la radio logicielle.

Puis durant la troisième semaine, j'ai eu l'occasion de rédiger une procédure d'installation de ces logiciels sur une machine virtuelle de l'IUT. Cette procédure pourra être utilisée plus tard pour le projet d'un autre étudiant par exemple.

Ensuite, lors de la quatrième semaine, j'ai pu tester ces logiciels tout en me penchant sur le fonctionnement du protocole utilisé par la télécommande.

Au cours de la cinquième semaine, je me suis initié à GNURadio et j'ai pu aussi rédiger des programmes pouvant être intégrés comme blocs à GNURadio.

Après avoir réussi à comprendre et à faire fonctionner le protocole de la télécommande avec la radio SDR, j'ai pu appliquer cette même démarche pour le protocole d'une station météo. L'étude du protocole et l'écriture des programmes m'ont pris deux semaines en tout.

Enfin, j'ai pu améliorer mes programmes, écrire des commentaires afin de faciliter leur compréhension et rédiger un support technique expliquant ma démarche et la façon d'arriver à mon résultat.

Conclusion

En conclusion, mon stage au sein du LIUPPA m'a permis de découvrir les protocoles utilisés en 433 MHz dans le domaine de l'IoT. J'ai pu analyser, décoder et rejouer des messages en étudiant la couche de télécommunication et la couche des données.

Grâce à ce stage, j'ai ainsi pu mettre en lumière l'aspect sécurité de l'IoT, car de nos jours ce genre de technologies est présent partout. Avec des outils comme le Flipper Zero, ces manipulations du signal sont accessibles à n'importe qui. Ces systèmes pourraient alors être massivement exploités pour des actions malveillantes, dans le but de voler des données confidentielles ou de modifier le comportement des appareils.

Ainsi, les constructeurs d'appareils IoT doivent désormais mettre l'accent sur la confidentialité et l'intégrité des données envoyées en utilisant des méthodes de chiffrement ou de signature selon le cas de figure.

Pour aller plus loin, on pourrait aussi étudier des appareils IoT utilisant du chiffrement, ou travailler sur des appareils utilisés en masse de nos jours, comme des télécommandes utilisant un système de code tournant, ce qui est utilisé pour ouvrir des portails ou des voitures.

Bilan Personnel

Durant mon stage au sein du LIUPPA, j'ai pu réaliser des activités qui m'ont permis de développer des compétences et des savoir-êtres.

Tout d'abord, j'ai fait une recherche bibliographique sur les logiciels SDR, ce qui m'a permis de découvrir des outils utilisés dans le cadre de la radio SDR et de comprendre leur fonctionnement. J'ai ensuite rédigé une procédure d'installation pour ces logiciels, ce qui m'a permis de développer mes compétences rédactionnelles et de synthèse.

J'ai également eu l'occasion de travailler sur l'étude d'une télécommande et d'une station météo en utilisant ces mêmes logiciels SDR. J'ai pu décoder les signaux émis par ces appareils et comprendre leur fonctionnement. Durant cette activité j'ai pu m'améliorer en analyse de signal et en télécommunication.

Enfin, j'ai dû rédiger des documents techniques et pédagogiques sur la radio SDR 433 MHz. Cette activité m'a offert l'occasion d'acquérir une démarche professionnelle de vulgarisation scientifique et de rédaction technique.

Ce stage m'a ainsi permis de développer des compétences techniques et professionnelles, mais, cela m'a également donné l'occasion de découvrir le vaste domaine de la radio logicielle et de l'IoT.

Définitions

IoT: Internet of Things, internet des objets

SDR: Software Defined Radio, radio logicielle

MLI: Modulation en Largeur d'Impulsion

Transposition: Décaler un signal d'une fréquence donnée du point de vue spectral

Décodage: Passage du signal carré reçu au signal de données binaire.

BCD: Binaire Codé Décimal, on code chaque chiffre d'un nombre décimal sur 4 bits.

Spoofing: Usurpation

CRC: Contrôle de Redondance Cyclique, c'est un outil de détection d'erreurs.

Flipper Zero: Couteau suisse des radiofréquences permettant par exemple d'enregistrer ou de rejouer des signaux en toute simplicité.

Protocole: Mise en forme des données en binaire.

Script: Petit programme écrit dans un langage interprété pour effectuer une tâche ou une série de tâches spécifiques.

Flowgraph: Diagramme de flux, représentation graphique d'un programme GNURadio

Code tournant: Système permettant de prévenir le forçage par ré-émission d'un signal capturé en générant un nouveau code pseudo-aléatoire à chaque émission.

Bibliographie

Présentation du LIUPPA

Organigramme du LIUPPA

URH, documentation

RTL_433, documentation

GNU Radio, tutoriels

Faire fonctionner RTL_433 sur Windows

Documentation de SoapySDR

Documentation technique du protocole de la télécommande

Code source du décodeur de la station météo pour RTL_433

Fonction de calcul du CRC utilisée par RTL_433

Annexes

Lien vers la page principale du dépôt git

Documents techniques rédigés

- Bibliographie des logiciels SDR
- Procédure d'installation des logiciels et tests
- Etude du capteur de température

Programmes écrits

- Programmes d'émission et de réception pour la télécommande
- Programmes d'émission et de réception pour la station météo

Résumé (Abstract)

Dans le cadre de mon stage au sein du LIUPPA (Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour), j'ai mené une étude approfondie sur les protocoles utilisés en 433 MHz dans le domaine de l'IoT. Ce sujet m'a permis d'en apprendre davantage sur les communications radio et les enjeux de sécurité liés à l'IoT.

Durant ce rapport de stage, j'étudie les communications radio des appareils fonctionnant dans la bande de fréquence 433 MHz, le décodage des données transmises, l'enregistrement et la retransmission des données.

Dans le cadre de cette étude, j'ai ainsi mis en lumière l'importance de la sécurité dans l'IoT, car ces technologies sont de plus en plus répandues et peuvent être facilement exploitées à des fins malveillantes.

Mots-clés : IoT, 433 MHz, communications radio, sécurité, SDR, expérimentation, programmation.

As part of my internship at LIUPPA (Laboratoire d'Informatique de l'Université de Pau et des Pays de l'Adour), I carried out an in-depth study of 433 MHz protocols used in the IoT field. This subject enabled me to learn more about radio communications and security issues related to the IoT.

In this internship report, I study the radio communications of devices operating in the 433 MHz frequency band, the decoding of transmitted data, and the recording and retransmission of data.

As part of this study, I have highlighted the importance of security in the IoT, as these technologies are becoming increasingly widespread and can be easily exploited for malicious purposes.

Keywords: IoT, 433 MHz, radio communications, security, SDR, experimentation, programming.