Analyse et sécurité d'une sonnette sans fil avec GNU Radio

Culture Sciences de l'Ingénieur

La Revue 3E.I

école normale supérieure paris-saclay

Thomas LAVARENNE[1]

Édité le 15/06/2026

[1] *Enseignant de Sciences Physiques en BTS CIEL option IR – Académie de Créteil.*

Cette ressource fait partie du N°120 de La Revue 3EI du 3ème trimestre 2026.

Après avoir étudié le filtrage et le traitement de signaux simulés (voir ressource « Filtrage numérique avec GNU Radio : approche expérimentale » [1]), il est intéressant de se confronter à des signaux radio réels issus de systèmes du quotidien. Cette approche permet d'introduire concrètement plusieurs notions fondamentales de la radio logicielle moderne : représentation complexe des signaux, traitement en bande de base, analyse fréquentielle et démodulation numérique.

Contrairement aux signaux purement réels manipulés habituellement en électronique ou en traitement analogique, les récepteurs SDR (Software Defined Radio) fournissent généralement des signaux complexes composés de deux voies : une composante en phase I et une composante en quadrature Q. Cette représentation, très présente dans les systèmes de communication modernes, permet de conserver simultanément les informations d'amplitude et de phase du signal reçu tout en simplifiant de nombreuses opérations de traitement numérique.

L'objectif de cet article est de proposer une approche expérimentale et progressive de ces notions à l'aide de GNU Radio. Cet environnement graphique permet de construire facilement des chaînes de traitement tout en visualisant en temps réel les différentes étapes du traitement du signal. Chaque opération peut ainsi être isolée, observée et reliée directement aux concepts théoriques associés.

Le support choisi ici est une sonnette sans fil fonctionnant dans la bande ISM 433 MHz. Ce type de dispositif utilise généralement une modulation simple de type OOK (On-Off Keying), particulièrement adaptée à une première approche de la démodulation radio. À partir du signal reçu par une clé SDR, nous montrerons comment observer le spectre, extraire l'enveloppe du signal, décoder les informations transmises puis reproduire l'émission à l'aide d'un émetteur SDR.

Au-delà de l'aspect purement technique, ces expériences permettent également d'aborder certaines problématiques concrètes liées à la robustesse et à la sécurité des communications radio : rejeu de trames, sensibilité au brouillage et limites de certains protocoles radio simples encore présents dans des équipements grand public.

1 - Réception d'un signal radio réel avec un dispositif SDR

Avant de rentrer dans le vif du sujet, il est nécessaire de comprendre la nature du signal fourni par un récepteur SDR, et en particulier sa représentation en bande de base sous forme complexe.

1.1 - Passage en bande de base

Lorsqu'un signal radio est capté par une clé SDR, celui-ci n'est pas directement fourni sous sa forme haute fréquence (par exemple 433,92 MHz). Le récepteur effectue en interne une opération de transposition fréquentielle : il sélectionne une fréquence centrale puis ramène le signal autour de cette fréquence vers la bande de base (0 Hz).

Autrement dit, si l'on règle le récepteur sur une fréquence f_c, le signal de sortie ne contient plus directement les composantes autour de f_c, mais les écarts de fréquence par rapport à cette valeur. Une composante située à $f_c+\Delta f$ apparaîtra ainsi comme une composante à $+\Delta f$ en bande de base.

Ce processus repose sur une multiplication par un signal sinusoïdal complexe (c'est-à-dire une voie en cosinus et une voie en sinus), suivie d'un filtrage passe-bas. Il permet de conserver toute l'information utile du signal tout en le ramenant dans une bande de fréquence exploitable numériquement.

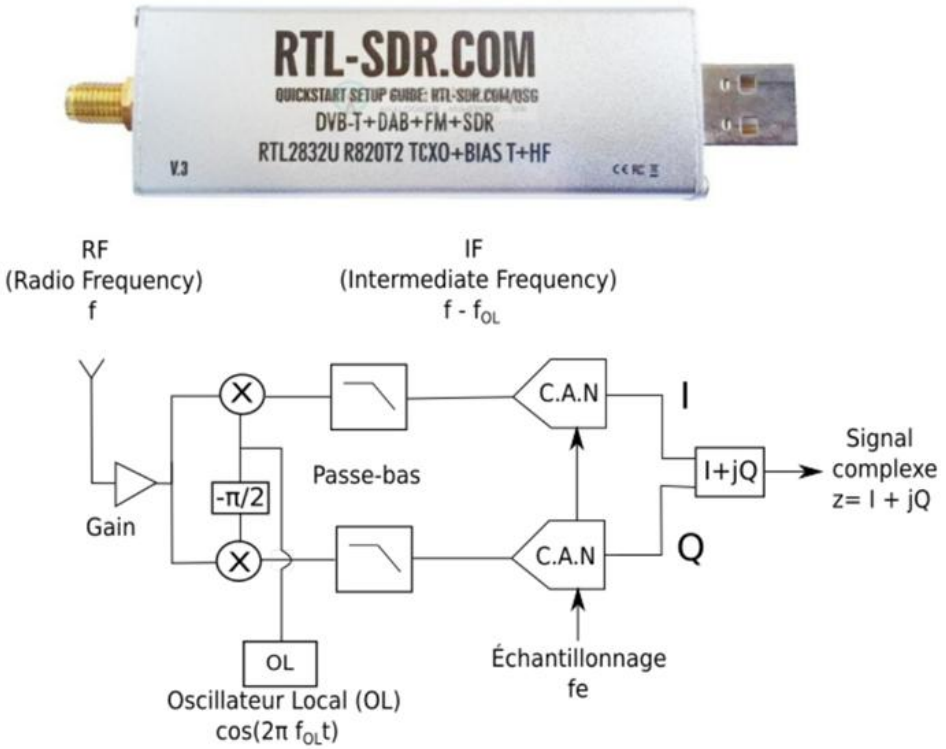


Figure 1 : Principe de fonctionnement d'un récepteur SDR (clé RTL-SDR)

Le signal fourni par un récepteur SDR est un signal complexe, constitué de deux composantes : I (en phase) et Q (en quadrature). Il s'écrit :

$$s(t) = I(t) + jQ(t)$$

Cette représentation permet de conserver à la fois l'amplitude et la phase du signal, ce qui est indispensable pour l'analyse des modulations. Dans GNU Radio, ces signaux sont manipulés sous forme de flux complexes (ports bleus), directement exploitables par les blocs de traitement.

1.2 - Visualisation du spectre

Pour observer le contenu fréquentiel du signal, on utilise le bloc *QT GUI Frequency Sink*. Celui-ci réalise une estimation du spectre à partir de FFT (Transformées de Fourier Rapides) calculées sur des blocs d'échantillons, ce qui permet une visualisation en temps réel proche d'un analyseur de spectre numérique. Dans le cas d'un signal complexe, le spectre s'étend de $-fe/2$ à $+fe/2$, où fe

est la fréquence d'échantillonnage. Le centre du spectre (0 Hz) correspond à la fréquence sur laquelle le récepteur est accordé :

- Les fréquences positives correspondent aux composantes au-dessus de la fréquence centrale ;
- Les fréquences négatives aux composantes situées en dessous.

Cette représentation permet de visualiser directement la position fréquentielle des signaux reçus. Nous allons maintenant appliquer ces principes à un exemple simple : la réception et la démodulation d'un signal de sonnette sans fil.

2 - Démodulation du signal reçu par une sonnette sans fil

Les sonnettes sans fil constituent un excellent support pédagogique pour introduire la démodulation de signaux radio. Leur fonctionnement repose généralement sur une modulation simple de type OOK (On-Off Keying), dans laquelle la présence ou l'absence de porteuse code l'information. L'objectif ici est de montrer comment extraire cette information à partir du signal complexe fourni par le récepteur SDR.

Figure 2 : Sonnette sans fil étudiée

2.1 - Observation du signal reçu

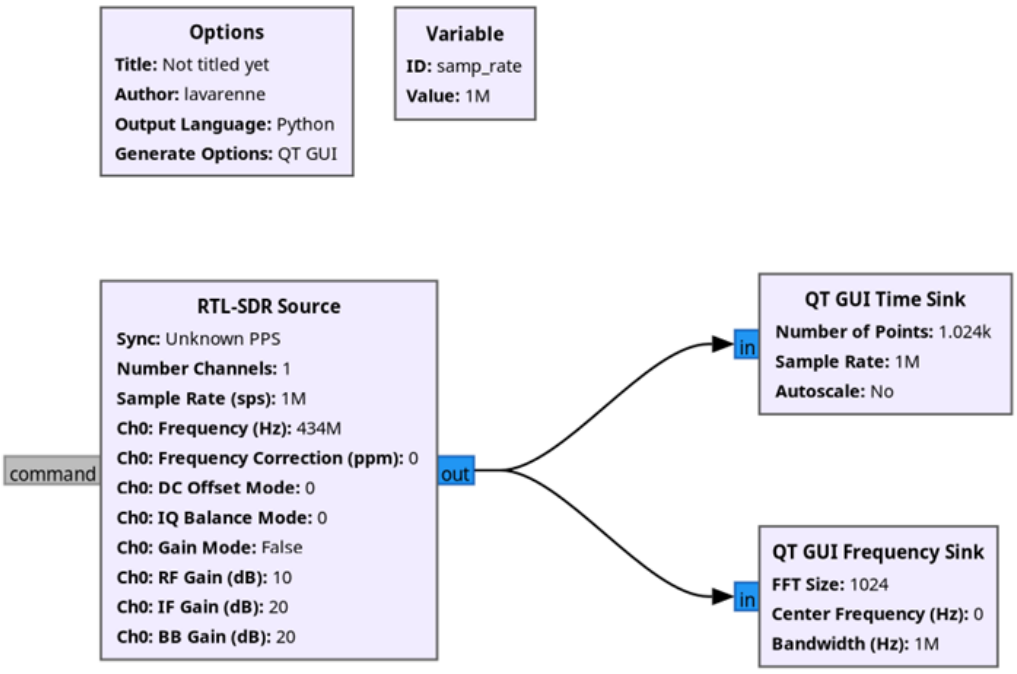


Figure 3 : Blocs GNU Radio pour afficher les représentations temporelles et fréquentielles

Après avoir réglé le récepteur sur la fréquence de la sonnette (typiquement autour de 433,92 MHz), l'appui sur le bouton émet un signal visible dans le spectre. Le paramètre **samp_rate** règle le taux d'échantillonnage qui est appliqué au signal ramené en bande de base. Il est important de le régler afin de ne pas être dans le cas d'un sous-échantillonnage ou d'un suréchantillonnage par rapport au signal utile transporté.

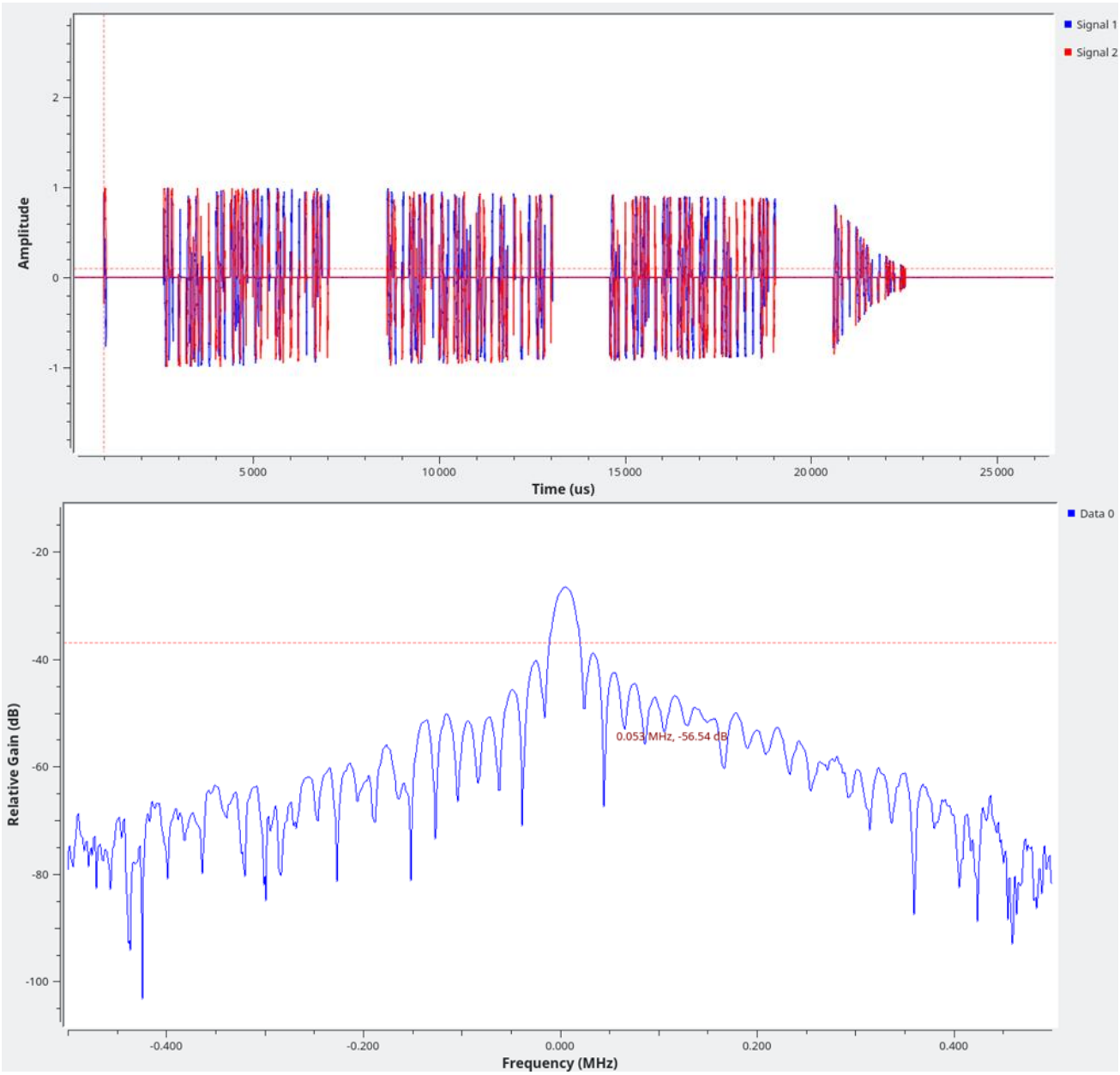


Figure 4 : Représentation temporelle et fréquentielle caractéristiques d'une modulation OOK

2.2 - Enregistrement et rejeu du signal

Pour plus de commodité, il est courant de réaliser une acquisition du signal émis avec un bloc File Sink (sous forme de fichier binaire *sonnette.bin* ici) et de le rejouer ensuite en boucle avec l'association des blocs File Source et Throttle (ce dernier étant nécessaire lorsqu'aucun périphérique matériel ne cadence l'échantillonnage).

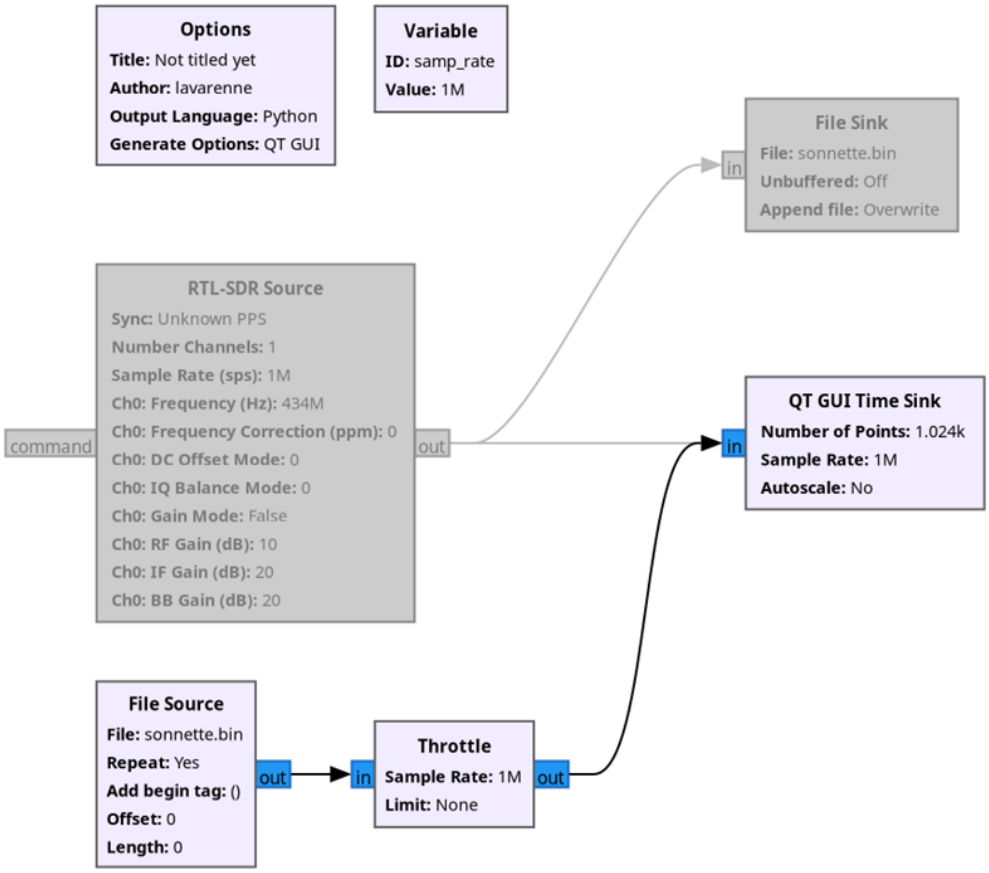


Figure 5 : Enregistrement puis rejeu du signal. En grisé, les blocs File Sink et RTL-SDR Source sont désactivés. Ils étaient actifs lors de l'acquisition

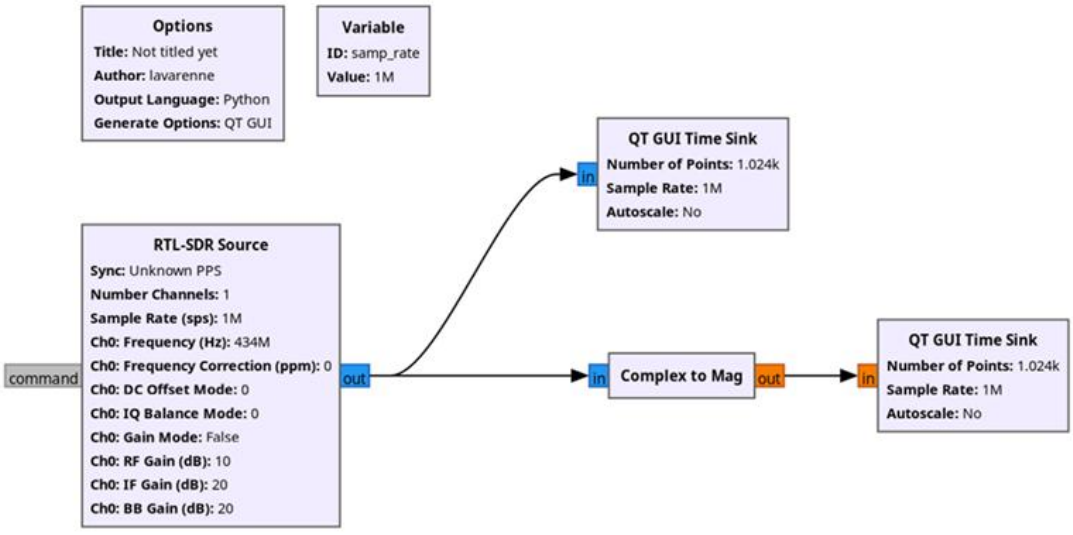
2.3 - Extraction de l'enveloppe

La modulation OOK repose sur la variation de l'amplitude du signal. L'information est donc contenue dans l'enveloppe du signal complexe.

Cette enveloppe peut être obtenue simplement à l'aide du bloc *Complex to Mag* qui calcule le module du signal complexe, c'est-à-dire :

$$|s(t)|=\sqrt{(I^2+Q^2)}$$

Le signal obtenu est réel et correspond à la puissance instantanée du signal reçu.



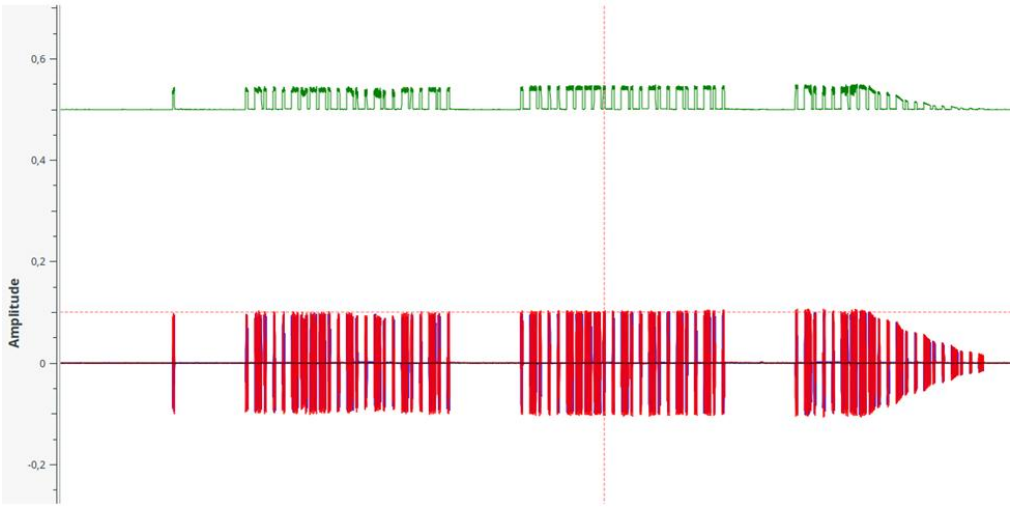


Figure 6 : Extraction de l'enveloppe avec le bloc Complex To Mag

2.4 - Embedded Python Block

Il peut être pédagogiquement intéressant de reconstruire cette opération d'extraction d'amplitude manuellement à l'aide des blocs GNU Radio :

- Un bloc **Complex To Float** pour séparer les deux voies I et Q ;
- Deux blocs **Multiply** pour calculer I^2 et Q^2 ;
- Un bloc **Add** ;
- Puis un **bloc Python personnalisé** pour effectuer la racine carrée.

Cette approche permet de relier explicitement les blocs au calcul mathématique. Pour commencer, réalisons le bloc *Complex To Mag²* qui réalise simplement $I^2 + Q^2$, sans effectuer la racine carrée :

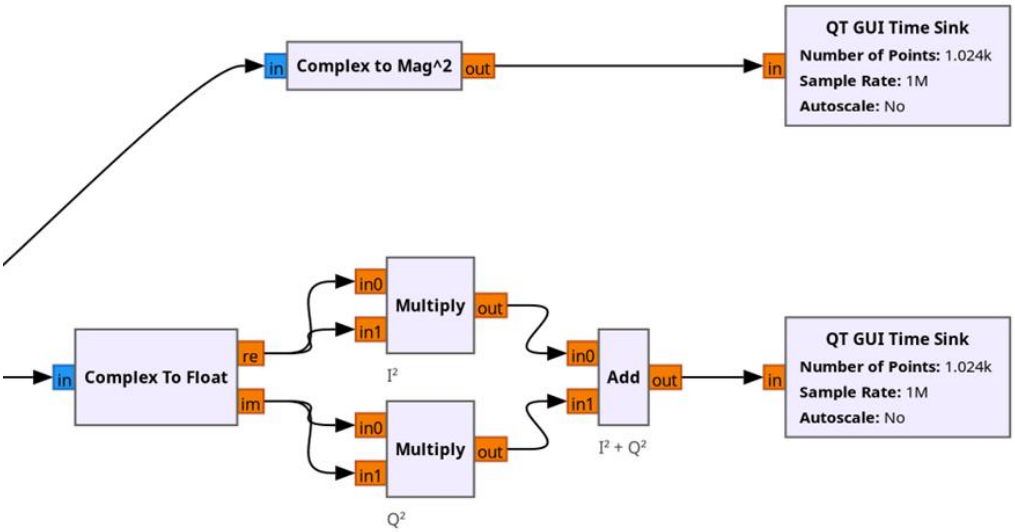


Figure 7 : Traitement GNU Radio pour obtenir $I^2 + Q^2$. Le bloc Complex To Float sépare les voies I et Q.

À ce stade, le signal est déjà démodulé et exploitable et le bloc *Complex To Mag²* est tout à fait utilisable en lieu et place du *Complex To Mag*. Néanmoins, c'est l'occasion de présenter une autre fonctionnalité de GNU Radio très intéressante : la possibilité d'implémenter soi-même ses propres blocs de traitement en Python.

En effet, il n'existe pas de blocs dans la librairie standard de GNU Radio qui applique l'opération racine carrée au flux d'échantillon. Qu'à cela ne tienne, nous allons donc le fabriquer nous-même !

Pour ce faire il existe des modèles de *Python Block* que nous allons utiliser et modifier pour réaliser l'opération voulue. Lorsque nous choisissons d'ajouter un tel bloc, nous avons la possibilité d'accéder au code source du bloc (*Open in editor*).

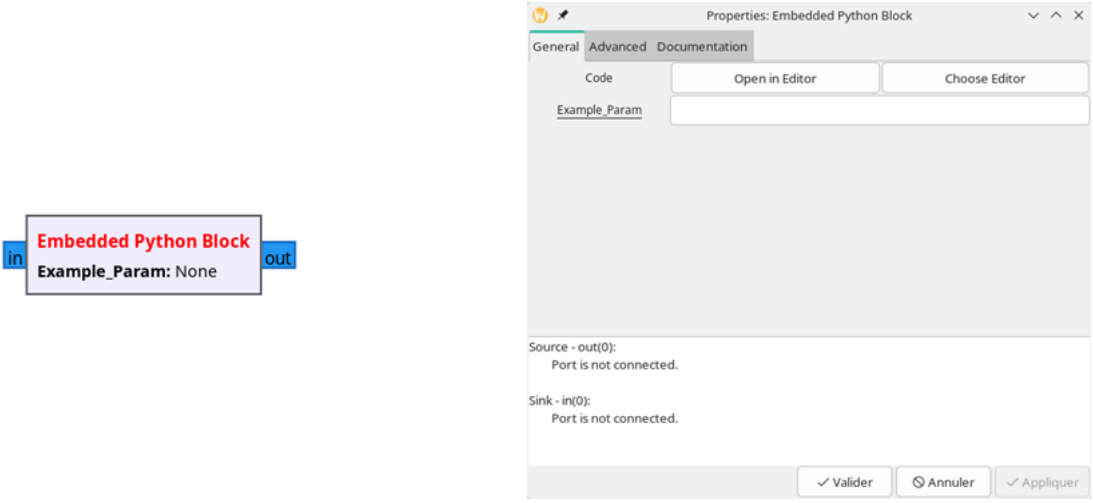


Figure 8 : Python bloc personnalisable

L'exemple qui s'ouvre par défaut est assez facile à comprendre. Il s'agit ici d'une simple multiplication par une constante. Adaptons donc cet exemple pour créer un bloc puissance qui prend en paramètre la valeur de la puissance à appliquer au flux (nous prendrons 0,5 pour appliquer la racine carrée).

```
"""
Embedded Python Blocks:

Each time this file is saved, GRC will instantiate the first
class it finds
to get ports and parameters of your block. The arguments to
__init__  will
be the parameters. All of them are required to have default
values!
"""

import numpy as np
from gnuradio import gr


class blk(gr.sync_block):  # other base classes are basic_block,
decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def __init__(self, example_param=1.0):  # only default
    arguments here
        """arguments to this function show up as parameters in
        GRC"""
        gr.sync_block.__init__(
            self,
            name='Embedded Python Block',   # will show up in GRC
            in_sig=[np.complex64],
            out_sig=[np.complex64]
        )
        # if an attribute with the same name as a parameter is
        found,
        # a callback is registered (properties work, too).
        self.example_param = example_param

    def work(self, input_items, output_items):
        """example: multiply with constant"""
        output_items[0][:] = input_items[0] * self.example_param
        return len(output_items[0])
```

```
"""
Embedded Python Blocks:

Each time this file is saved, GRC will instantiate the first
class it finds
to get ports and parameters of your block. The arguments to
__init__  will
be the parameters. All of them are required to have default
values!
"""

import numpy as np
from gnuradio import gr


class blk(gr.sync_block):  # other base classes are basic_block,
decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def __init__(self, puissance=1.0):  # only default arguments
    here
        """arguments to this function show up as parameters in
        GRC"""
        gr.sync_block.__init__(
            self,
            name='Puissance',   # will show up in GRC
            in_sig=[np.float32],
            out_sig=[np.float32]
        )
        # if an attribute with the same name as a parameter is
        found,
        # a callback is registered (properties work, too).
        self.puissance = puissance

    def work(self, input_items, output_items):
        """example: multiply with constant"""
        output_items[0][:] = input_items[0] * self.puissance
        return len(output_items[0])
```

Figure 9 : À gauche le bloc Python exemple qui multiplie tous les échantillons d'entrée par une constante. À droite, le bloc modifie pour appliquer une puissance à tous les échantillons d'entrée.

Sans entrer trop dans les détails expliquons les grandes lignes de ce traitement : le flux d'échantillon étant continu, l'ordonnanceur GNU Radio décide d'envoyer en entrée de chaque bloc un nombre fini d'échantillon (les **input_items**) à chaque appel de la méthode **work()** qui réalise le traitement souhaité et produit les échantillons de sortie (les **output_items**) si au moins une sortie est déclarée dans le constructeur. Mettons en évidence les parties modifiées afin de réaliser le traitement souhaité :

- Dans le constructeur, **example_param** change de nom et devient **puissance** ;
- Le nom du bloc passe de « **Embedded Python Block** » à « **puissance** » ;
- Attention aux types d'entrées et sorties (in_sig et out_sig) : dans l'exemple le type est **np.complex64,** nous les passons en **np.float32** ;
- **self.example_param = example_param** devient **self.puissance = puissance** afin de correspondre à ce que l'on souhaite effectuer ;
- Dans la méthode **work(),** appelée régulièrement par l'ordonnanceur de GNU Radio (tant qu'il y a des échantillons d'entrée) la ligne permettant de calculer les échantillons de sortie est modifiée : **output_items[0][:] = input_items[0]**self.puissance** ou ** représente l'opérateur puissance en Python.

Nous n'avons donc plus qu'à ajouter ce bloc à la suite du traitement et à appliquer une puissance 0.5 (racine carrée). Le résultat obtenu, présenté sur la figure ci-dessous, est identique en tous points à celui fourni par le bloc *Complex to Mag*.

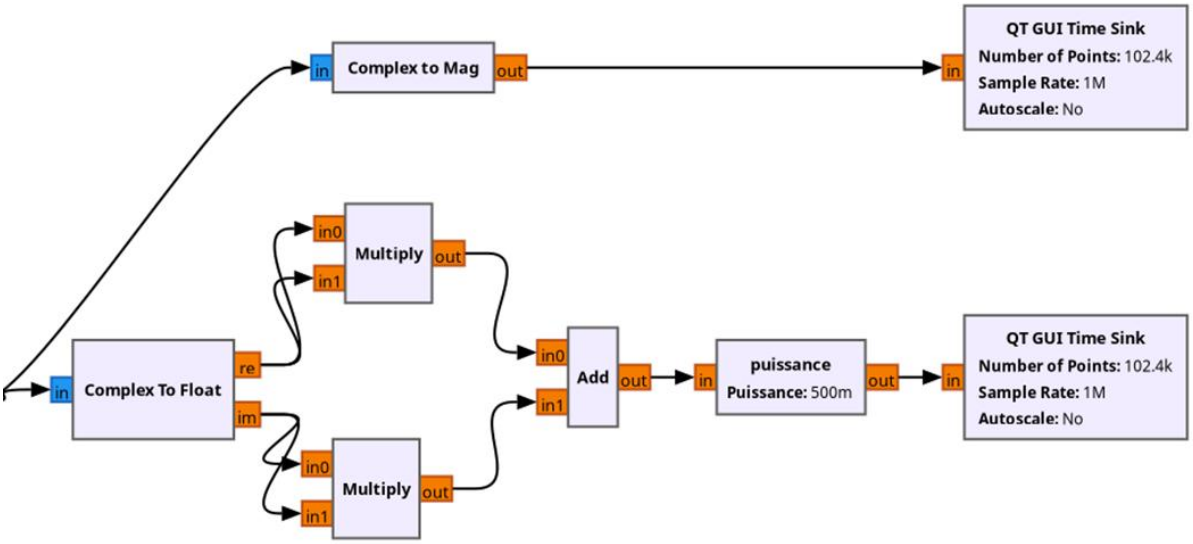


Figure 10 : Structure complète permettant de reproduire le traitement du bloc Complex To Mag

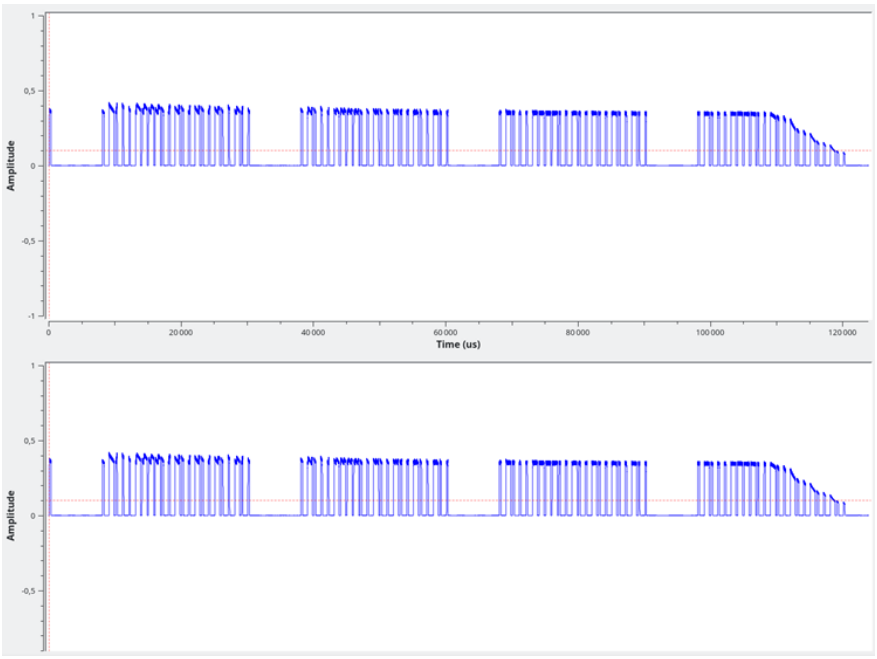


Figure 11 : Comparaison du résultat de notre traitement avec celui du Complex To Mag. Les deux signaux sont strictement identiques.

De cette manière on montre que les blocs de GNU Radio ne sont pas de simples « boîtes noires ». Leur fonctionnement repose sur des opérations mathématiques plus ou moins simples, que l'on peut reconstruire et comprendre pas à pas. Cette démarche permet de relier directement les outils logiciels aux concepts théoriques du traitement du signal, et contribue ainsi à une meilleure appropriation des notions étudiées.

3 - Analyse du signal et prolongements

3.1 - Décodage et caractérisation du signal

Une fois l'enveloppe extraite, le signal obtenu peut être observé dans le domaine temporel afin d'identifier la structure des trames émises par la sonnette. On distingue alors une succession d'impulsions correspondant aux bits transmis. La durée des états hauts et bas permet généralement de retrouver le codage utilisé par le constructeur.

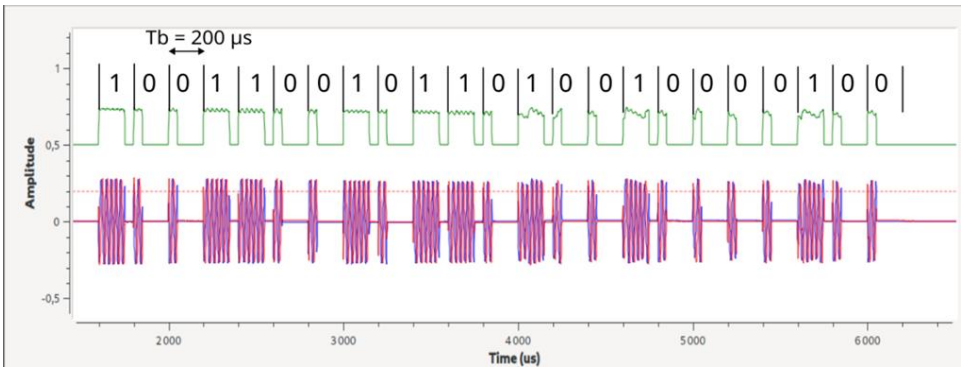


Figure 12 : Décodage du signal. Les informations binaires sont encodées en Pulse Length Encoding : un « 1 » correspond à un état haut de 150 µs puis un état bas de 50 µs et inversement pour le « 0 ».

Dans de nombreux systèmes simples de télécommande radio, l'information est transmise à l'aide de codages élémentaires reposant sur des variations de durée d'impulsion (PLE, Pulse Length Encoding) ou sur des transitions d'état.

GNU Radio permet alors d'aller plus loin en ajoutant des blocs de seuillage, de synchronisation ou encore de décodage personnalisé afin de reconstruire directement la suite binaire transmise.

3.2 - Reproduction et rejeu

Une fois le signal enregistré, il est possible de le rejouer à l'identique à l'aide d'un émetteur SDR compatible émission (HackRF One, USRP B206 mini-i, Pluto SDR, …).

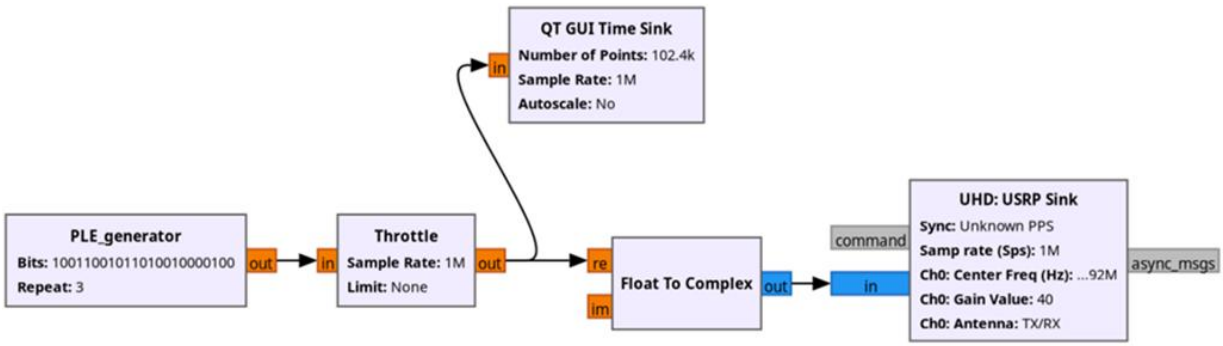
Figure 13 : À gauche : usrp b206 mini-i, au milieu : HackRF One, à droite : Pluto SDR. Trois dispositifs SDR compatibles émission.

Avertissement : *même dans la bande libre ISM 433, l'émission radiofréquence est strictement contrôlée. Avant de tenter une quelconque expérience de diffusion, assurez-vous d'être dans des conditions respectant les normes de l'ANFR (Agence Nationale des Fréquences) afin de ne perturber aucun système en fonctionnement au voisinage.*

La création d'un **Embedded Python Block** permettant de générer le signal en fonction de la suite binaire est un bon exercice. On déclare ainsi un nouveau bloc sans aucune entrée et avec une seule sortie (np.float32) qui génère le signal via deux fonctions **set_one(self)** et **set_zero(self)** puis l'envoie vers la sortie grâce à la fonction **work()**.

```
import numpy as np
from gnuradio import gr

class blk(gr.sync_block):
    def __init__(self, bits="10011001011010010000100", repeat=3):
        gr.sync_block.__init__(
            self,
            name='PLE_generator',
            in_sig=None,
            out_sig=[np.float32])
        self.bits = bits
        self.repeat = repeat
        self.signal = []

        for _ in range(repeat):
            for b in bits:
                if b == "1":
                    self.set_one()
                else:
                    self.set_zero()

            self.signal += [0]*1000    # espace entre répétitions
        self.signal += [0]*20000      # espace entre trames
        self.signal = np.array(self.signal, dtype=np.float32)
        self.index = 0

    def set_one(self):          # HIGH 150 us + LOW 50 us
        self.signal += [1]*150
        self.signal += [0]*50

    def set_zero(self):         # HIGH 50 us + LOW 150 us
        self.signal += [1]*50
        self.signal += [0]*150

    def work(self, input_items, output_items):
        out = output_items[0]
        for i in range(len(out)):
            out[i] = self.signal[self.index]
            self.index += 1
            if self.index >= len(self.signal):
                self.index = 0
        return len(out)
```

Figure 14 : Bloc permettant de générer le signal dans GNU Radio



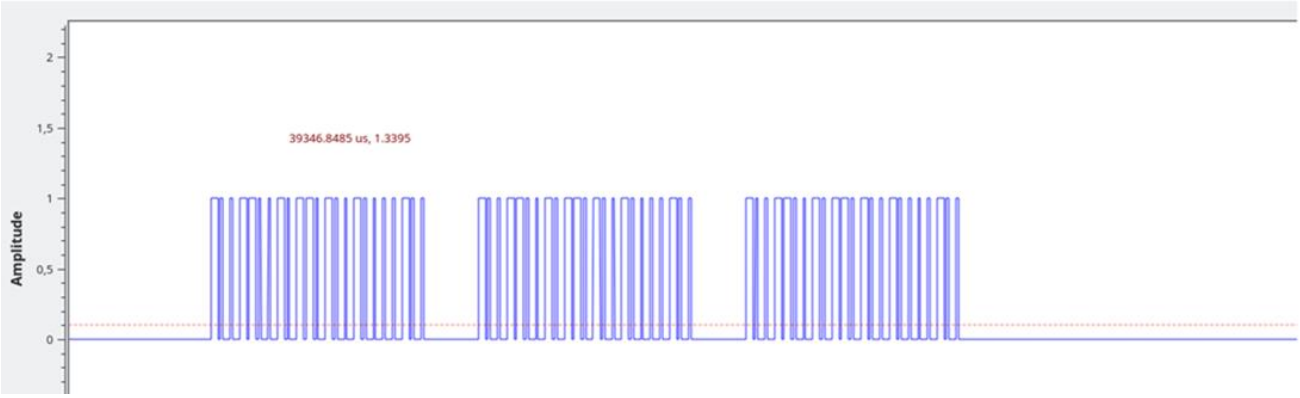


Figure 15 : Émission du signal avec un USRP. En haut, la structure permettant d'envoyer le signal sur l'usrp. En bas, le résultat observer dans le Time Sink.

Cette manipulation met en évidence le principe des attaques par rejeu (« replay attack »), consistant à réémettre un signal légitime précédemment capturé. Dans le cas de systèmes simples ne mettant en œuvre aucun mécanisme cryptographique ou code tournant (« rolling code »), cette réémission peut suffire à reproduire l'action d'origine. Cette expérience constitue une introduction intéressante aux problématiques de cybersécurité appliquées aux systèmes radiofréquences.

3.3 - Brouillage du système

Il est également possible d'étudier expérimentalement la sensibilité du système au brouillage radio.

Figure 16 : Principe du brouillage : on approche un HackRF émettant dans la bande ISM 433

Pour générer le bruit large bande, on utilise GNU Radio et le bloc Noise Source et on fait passer le flux dans un filtre passe-bas comme indiqué sur la figure 17.

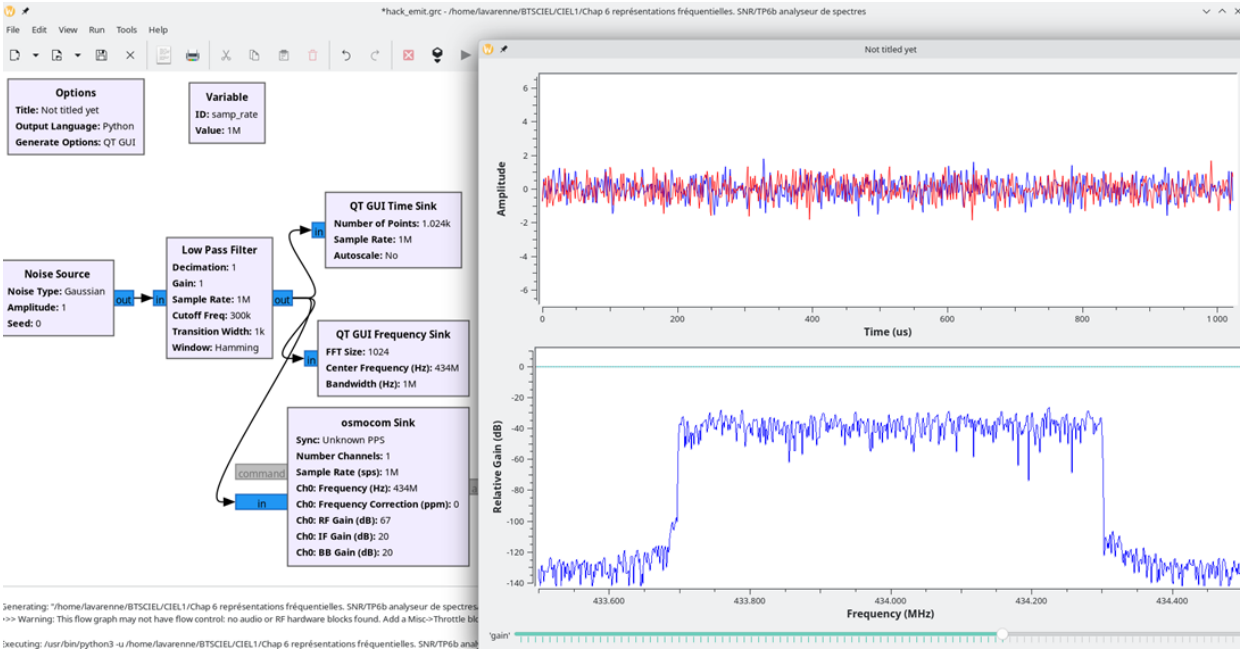


Figure 17 : Génération du bruit large bande et émission avec le HackRF (bloc Osmocom Sink).

Un analyseur de spectre peut être utile pour visualiser les niveaux avec et sans brouillage. Sur la figure 18 ci-dessous on constate bien que le signal du bouton de la sonnette est complètement noyé dans le bruit généré par le HackRF.

L'ajout d'un signal parasite dans la bande ISM 433 MHz dégrade rapidement la réception et peut empêcher complètement le décodage des trames de la sonnette. Ces expériences permettent d'illustrer concrètement l'importance du rapport signal sur bruit ainsi que la vulnérabilité potentielle des systèmes radio simples face aux perturbations volontaires ou non.

Bien entendu, le brouillage d'une simple sonnette sans fil présente ici surtout un intérêt pédagogique. Néanmoins, cette expérience met en évidence une problématique bien réelle : certaines alarmes domestiques bas de gamme, détecteurs sans fil ou automatismes utilisent encore des protocoles proches de ceux étudiés ici. Dans ce contexte, des dispositifs SDR peu coûteux associés à un générateur de bruit peuvent suffire à perturber localement les communications radio entre les différents équipements.

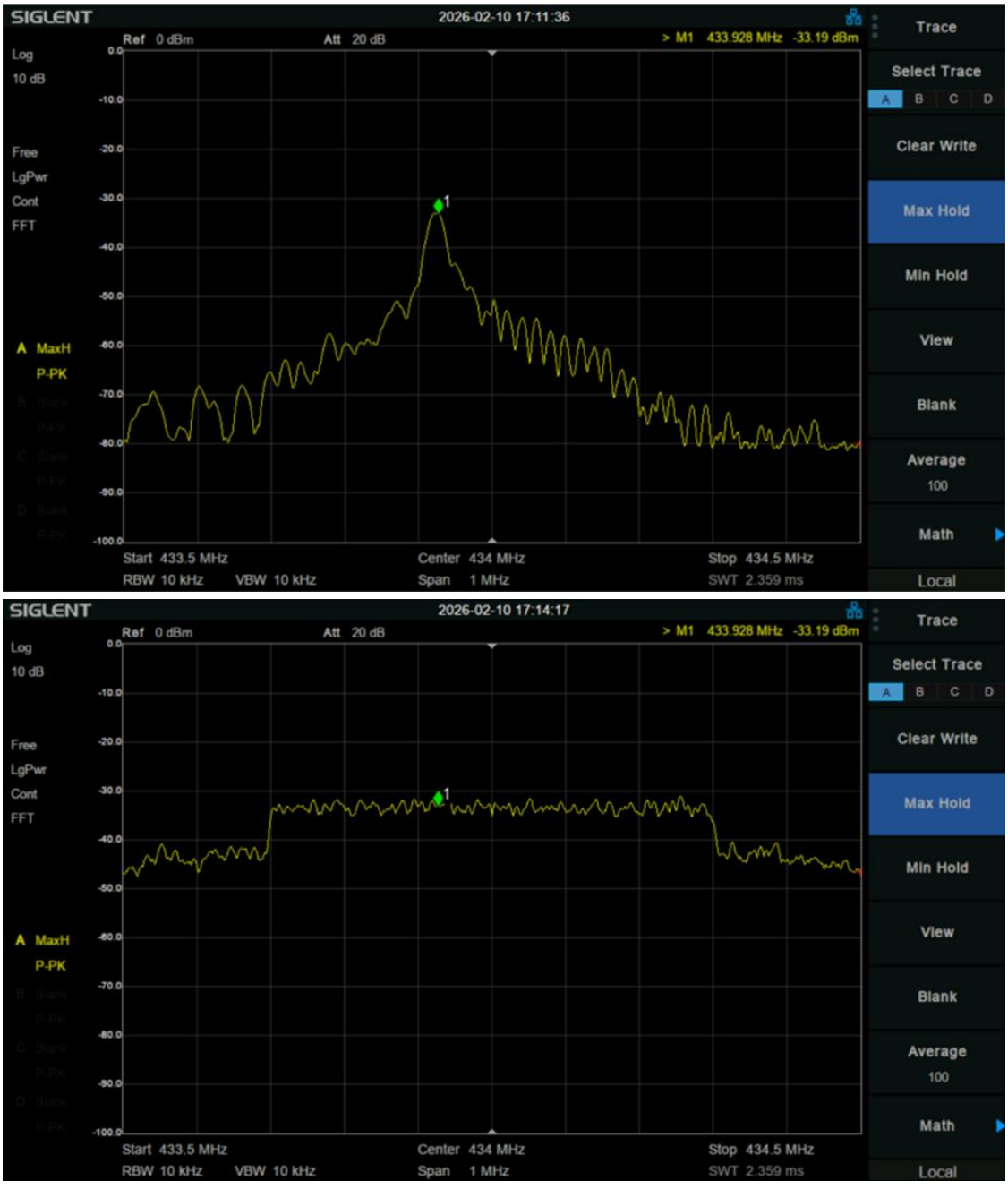


Figure 18 : Visualisation sur l'analyseur de spectre à proximité de la sonnette : en haut, sans brouillage, en bas avec le brouillage activé.

Les systèmes modernes cherchent généralement à se prémunir contre ces vulnérabilités grâce à différentes solutions :

- Détection de brouillage radio ;
- Changement dynamique de fréquence ;
- Redondance des transmissions ;
- Augmentation de la robustesse des protocoles ;
- Ou encore utilisation de liaisons filaires pour les fonctions critiques.

L'objectif de cette manipulation n'est donc évidemment pas de fournir une méthode d'attaque, mais de sensibiliser aux limites de certains systèmes radio simples et à l'importance de concevoir des architectures de communication robustes face aux perturbations intentionnelles ou non.

4 - Conclusion

L'utilisation de GNU Radio associée à un récepteur SDR permet de mettre en œuvre rapidement des expériences concrètes de réception et de démodulation radio. À travers l'exemple simple d'une sonnette sans fil, il devient possible d'introduire progressivement plusieurs notions fondamentales du traitement numérique du signal : représentation complexe I/Q, transposition en bande de base, analyse fréquentielle, extraction d'enveloppe et démodulation.

L'approche graphique de GNU Radio présente un intérêt pédagogique particulièrement important : chaque étape du traitement peut être isolée, visualisée et modifiée simplement, ce qui facilite fortement la compréhension des phénomènes étudiés et permet de relier directement les concepts théoriques aux signaux réels. Les expériences proposées montrent également qu'il est relativement simple, à l'aide de matériels SDR désormais accessibles, d'analyser, reproduire ou perturber certaines communications radio grand public. Ces manipulations permettent ainsi d'aborder concrètement plusieurs problématiques modernes liées à la robustesse et à la sécurité des transmissions sans fil. Au-delà de cet exemple volontairement simple, les mêmes principes se retrouvent dans de nombreux systèmes modernes de communication radio. Cette approche constitue ainsi une excellente porte d'entrée vers l'étude plus avancée des modulations numériques, des protocoles radio et de la radio logicielle.

Référence

[1]: Filtrage numérique avec GNU Radio : approche expérimentale, T. Lavarenne, juin 2026, https://sti.eduscol.education.fr/si-ens-paris-saclay/ressources_pedagogiques/filtrage-numerique-avec-gnu-radio-approche-experimentale