Le récepteur de radio logicielle RSP1 : 8 MHz de bande passante pour moins de 20 euros

Jean-Michel Friedt, FEMTO-ST temps-fréquence, 20 août 2022

Deux composants de la société anglaise Mirics ont été associés pour proposer le RSP1, une radio logicielle à moins de 20 euros couvrant 10 kHz–2 GHz (donc de la bande HF, notamment des radioamateurs, à GPS et Iridium) mais surtout offrant une bande passante jusqu'à 8 MHz. Malheureusement son utilisation s'appuie sur la bibliothèque propriétaire SDRPlay dont l'installation n'est pas de tout repos. Nous réussirons tout de même à recevoir un signal de transfert de temps issu d'un satellite géostationnaire et le décoder.

1 RSP1 : une alternative au RTL-SDR

Alors que nous cherchions à réapprovisionner les stocks de RTL-SDR en prévision de l'année d'enseignement à venir, force est de constater que l'approvisionnement des récepteurs munis de récepteurs radio R820T2 s'est tari et les prix avec frais de port se sont envolés. Nous nous sommes laissés par conséquent attirer par les publicités d'AliExpress qui vantaient les mérites du RSP1, qui en plus de proposer un prix abordable (notamment dans le cadre d'un enseignement universitaire), répond à une ambition d'augmenter la bande passante de mesure accessible au-delà des 2,4 MHz des vénérables RTL-SDR, insuffisante pour démoduler le signal échangé par satellite géostationnaire de transfert de temps et fréquence qui nous intéresse par ailleurs [1]. Ce protocole de communication impose au moins 5 MHz de bande passante, et trouver un récepteur de radio logicielle accessible mais suffisamment rapide est nécessaire pour permettre à l'auditoire éclairé de reproduire les mesures proposées.

Nous découvrirons [2, 3] que ce RSP1 est fortement inspiré du SDRPlay anglais [4] en alliant deux composants dédiés à la réception de radio logicielle, les MSi001 et MSi2500. Ces deux composants, conçus et produits par la société anglaise Mirics, jouent des rôles quasiment identiques aux R820T2 et RTL2832U des récepteurs RTL-SDR, avec une séparation identique de la réception du signal radiofréquence et transposition en bande de base par le premier, et conversion du flux IQ en bande de base vers un flux numérique sur bus USB par le second. Parmi les bénéfices du récepteur de radio logicielle RSP1, une couverture de bandes de fréquences inférieures à celles accessibles par RTL-SDR ou même les AD936x qui équipent la PlutoSDR ou l'Ettus Research B210 pour descendre sous le MHz pour le plus grand plaisir des radioamateurs encore utilisateurs des bandes Haute-Fréquence (HF, inférieure à 30 MHz), mais surtout une bande passante allant jusqu'à 8 MHz ou environ 4 fois plus que le RTL-SDR qui motive cette étude.

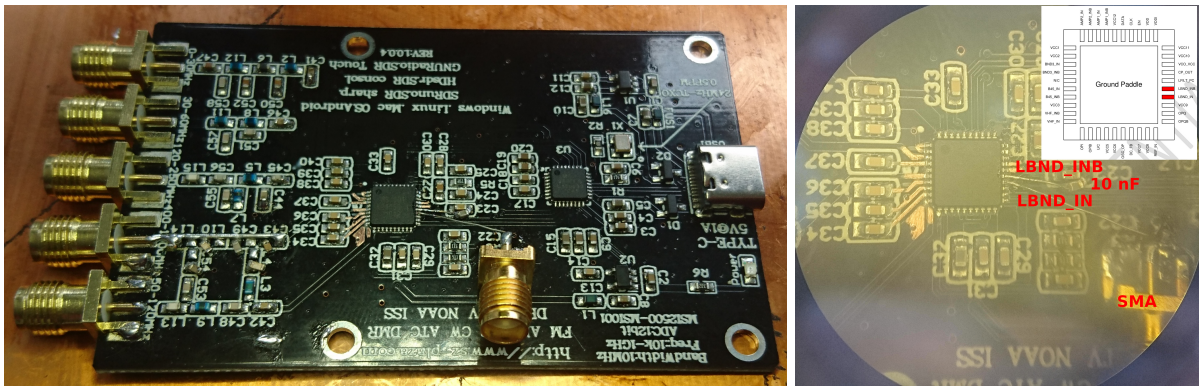

FIGURE 1 – Ajout de la connexion à l'entrée en bande L du MSi001 : gauche vue d'ensemble, droite gros plan sur les deux broches à connecter au connecteur SMA et à un condensateur de 10 nF à la masse. Le vernis des pistes autour du circuit intégré a volontairement été gratté pour mieux visualiser les pistes allant aux divers filtres passe-bande menant à chaque connecteur d'antenne. En insert le brochage extrait de la documentation technique du MSi001.

Le circuit (Fig. 1, gauche) présente l'originalité de fournir une multitude de connecteurs radiofréquences

d'antennes suivis de filtres passe bande dédiés à chaque bande de fréquence : en effet, le MSi001 est muni de 6 entrées radiofréquences ajustées chacune pour une bande bien spécifique. AliExpress propose deux déclinaisons du RSP1, l'une annoncée fonctionnant entre 10 kHz et 1 GHz, l'autre entre 10 kHz et 2 GHz.

Lors de l'achat, nous avions décidé d'économiser une paire d'euros en sélectionnant le modèle couvrant 10 kHz–1 GHz, nous disant qu'à composant égal il n'y avait pas de raison de ne pas pouvoir l'utiliser comme les modèles annoncés à 2 GHz, quitte à modifier un peu le matériel pour ajuster les différences entre les deux modèles qui s'appuient sur les mêmes circuits intégrés. La différence est significative car nous décrivons dans [1] pourquoi le signal qui nous intéresse dans le transfert de temps par satellite géostationnaire se trouve à 1,204 GHz, donc au-dessus des 1 GHz du modèle le moins cher.

Inutile de faire durer le suspens sur la différence entre les deux modèles : l'entrée 1–2 GHz n'est simplement pas routée sur les modèles annoncés à 1 GHz, et programmer le récepteur au-delà de 1 GHz se traduit par la perte totale de réception, le MSi001 commutant sur l'entrée en bande L (nomenclature IEEE de la bande de fréquences entre 1 et 2 GHz que nous retrouvons dans la documentation technique du composant en Fig. 1, droite) qui n'est pas branchée. Ce défaut se corrige facilement (Fig. 1) pour tout possesseur d'un fer à souder à panne un peu fine et d'une binoculaire : l'entrée LBND_IN (broche 24) est reliée à un connecteur radiofréquence (ici SMA) et l'entrée LBND_INB (broche 25, toutes deux à droite de la photographie de droite de Fig. 1) à un condensateur 10 nF relié à la masse. On pourra évidemment améliorer cette connexion triviale avec un filtre passe-bande en T ou Pi comme en sont munies les autres entrées. Nous constatons en effet que ce routage est effectué sur la version "full" du RSP1 couvrant la bande radiofréquence jusqu'en bande L à 2 GHz, mais que ce montage fait passer le signal radiofréquence par des DIP switches dont aucune datasheet ne mentionne les performances aux radiofréquences, et une piste bien longue pour un signal de 15 cm de longueur d'onde dans l'air (Fig. 2) longe la moitié supérieure du circuit imprimé. Ici encore, une multitude de filtres LC sont sensés ne conserver sur chaque entrée du MSi001 que la bande spectrale qui la concerne. Mais quelles sont les caractéristiques de ces filtres ?

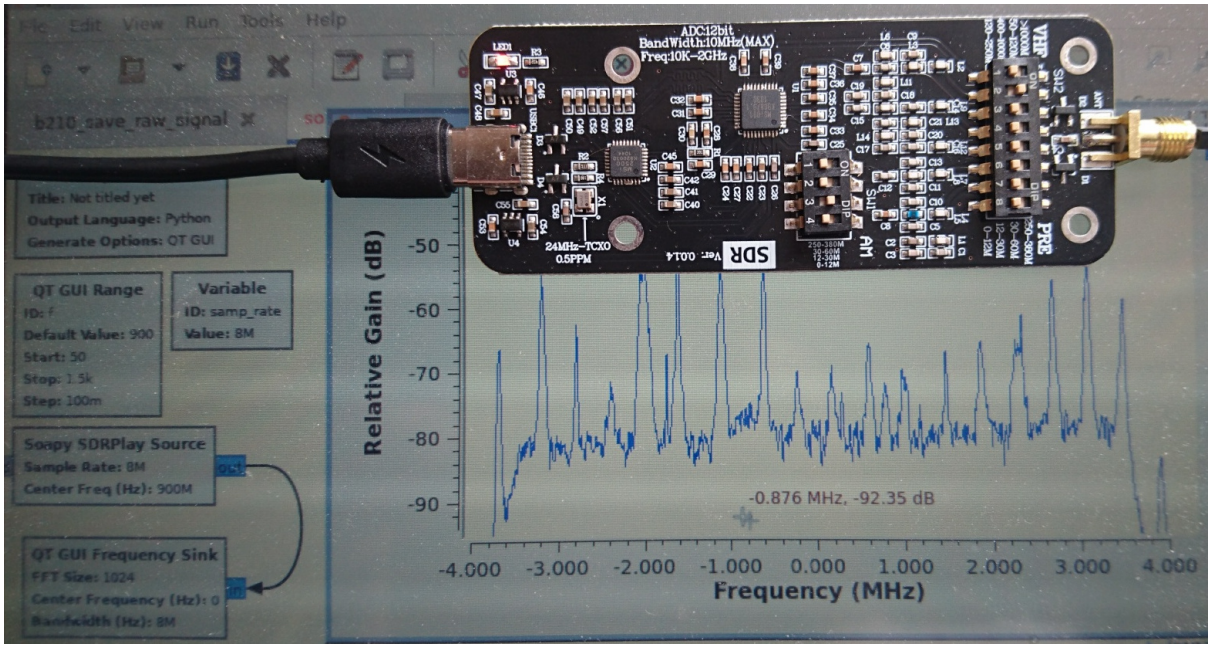

FIGURE 2 – La version "full" de RSP1 inclut un unique connecteur SMA mais une série de DIP switches permettant de router les signaux arrivant à l'antenne, incluant entre 1 et 2 GHz, vers la bonne broche du MSi001, au prix du passage des signaux radiofréquences dans des DIP switches et une piste micro-onde qui longe toute la longueur du circuit imprimé avant d'arriver aux broches LBND du "mauvais" coté du MSi001. En arrière plan, la bande FM centrée sur 98 MHz sur une bande passante de 8 MHz reçue à Paris, avec 18 stations observées simultanément.

2 Caractérisation des filtres radiofréquences

Notre première interrogation à la réception de la carte était de connaître la fonction de transfert des filtres passe bande situés entre les divers connecteurs SMA et les broches du MSi001 chargées de recevoir les signaux radiofréquences dans chaque bande, et ce notamment dans l'idée initiale d'utiliser la bande la plus haute en fréquence au-delà de 1 GHz avant d'avoir compris que la bande L est traitée par une broche complètement différente. La Fig. 3 fournit le module du coefficient de transmission de chaque filtre passe-bande dans l'ordre des connecteurs SMA sur le bord du circuit imprimé. Nous constatons que le filtre de la bande annoncée à 1 GHz commence à couper bien avant sa borne supérieure, pour atténuer d'une quinzaine de dB un signal incident à 1 GHz. Le cas du signal à 1,2 GHz n'est pas abordé ici puisque programmer le MSi001 pour une fréquence supérieure à 1 GHz fait simplement disparaître le signal entrant sur le connecteur SMA nommé 400–1000 MHz lorsque le MSi001 commute sur une autre broche d'entré qui n'était pas câblée.

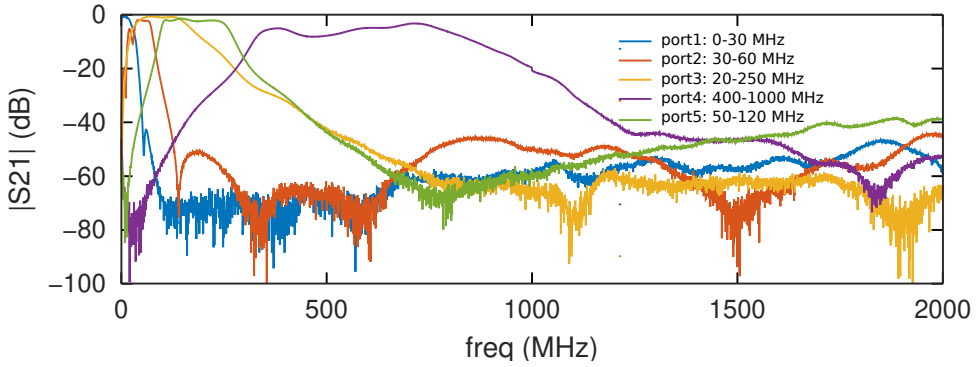

FIGURE 3 – Caractérisation en transmission (S_{21}) des filtres passe-bande placés entre chaque connecteur d'antenne et le récepteur radiofréquence MSi001.

Les filtres sont un peu mous, comme prévu par une implémentation en condensateur et inductance discrets pour économiser sur le coût de fabrication, et en particulier les pertes sur la bande la plus haute en fréquence sont relativement élevées. Ce constat correspond à l'utilisation souvent préconisée pour ce récepteur par les radioamateurs désireux de travailler dans les bandes de fréquence les plus basses plutôt que profiter de la fréquence d'échantillonnage élevée qui est plutôt bénéfique sur la bande des porteuses les plus élevées.

3 Logiciels de réception : GNU Radio `Soapy SDRPlay`

Étant confiants sur l'adéquation du matériel aux besoins de réception des liaisons satellitaires portant les informations de temps et fréquence échangés entre Observatoires européens (voir article [1] dans ce numéro), à savoir donc une porteuse autour de 1,204 GHz et une bande passante de mesure supérieure à 5 MHz, il nous faut aborder la partie logicielle.

Les choses sont ici beaucoup moins radieuses que lors de l'analyse du matériel. Le clone chinois de SDRPlay s'appuie sur la bibliothèque anglaise du même nom ... propriétaire et disponible uniquement en format binaire, fort heureusement pour GNU Linux x86 et ARM tout de même. La bibliothèque libre `libmirisdr` fournit une alternative libre qui n'a pas atteint le même niveau de maturité, pour ne simplement avoir jamais fonctionné au cours de nos essais. Même pour la bibliothèque propriétaire, la disponibilité du script `restartService.sh` pour relancer le démon `sdrplay_apiService` n'est pas anodine puisque nous y ferons appel à chaque lancement de GNU Radio pour espérer utiliser notre récepteur à 20 euros qui ne sera souvent pas détecé sinon.

Nous commençons par interdire le chargement des pilotes du noyau Linux chargés de contrôler les interface MSi001 et MSi2500 par l'ajout d'un fichier d'extension `.conf` dans `/etc/modprobe.d` contenant `blacklist msi001` et `blacklist msi2500`. Si le RPS1 a déjà été inséré dans un port USB, retirer ces modules du noyau lors des compilations qui vont suivre.

La bibliothèque et les divers outils de développement se téléchargent à `www.sdrplay.com` en indiquant le système d'exploitation et RSP1 comme modèle. Comme à chaque interaction avec une bibliothèque propriétaire dans un environnement libre, les chances de casser l'installation de son infrastructure de radio logicielle sont grandes et les essais qui se sont conclus par une procédure convaincante ne font pas exception à la règle, la multitude de couches intermédiaires n'aidant pas à retrouver les dépendances au premier abord. L'archive `SDRplay_Linux_Scripts_v0.2.zip` (l'extension `.zip` est de bien mauvaise augure) obtenue auprès de `www.sdrplay.com` contient deux scripts d'installation qui nous seront utiles, `1installAPI.sh` et `2buildSoapy.sh`. Le premier installera une bibliothèque propriétaire dans `/usr/local/lib` et le second recompilera SoapySDR pour lui ajouter le module `sdrplay` nécessaire à faire le lien avec GNU Radio. Le second script va poser problème pour interagir avec la distribution binaire de GNU Radio fournie par Debian/GNU Linux et doit être modifié : en effet, par défaut son comportement consiste à cloner le dépôt GitHub de SoapySDR, de le configurer et le compiler. Ce faisant, la branche de développement sera compilée tel qu'en atteste

```
$ ldd Dev/SoapySDR/build/apps/SoapySDRUtil
  linux-vdso.so.1 (0x00007fff66591000)
  libSoapySDR.so.0.8-2 => /home/jmfriedt/Dev/SoapySDR/build/lib/libSoapySDR.so.0.8-2
...
```

qui indique des bibliothèques d'extension `0.8-2` et on vérifiera que `which SoapySDRUtil` indique bien `/usr/local/bin/SoapySDRUtil` et non le binaire de la distribution éventuellement installé dans `/usr/bin`.

Le problème est que SoapySDR vérifie la cohérence de ses modules par un numéro d'ABI. Ainsi, Jeff Long, mainteneur de la version "stable" de GNU Radio (pour autant qu'elle existe) a corrigé la documentation de `https://wiki.gnuradio.org/index.php/Soapy#Versions` pour désormais indiquer non pas "*If GNU Radio is built against SoapySDR 0.8.0, then any module built against 0.8.x will load.*" mais préciser "*SoapySDR driver modules will be loaded only if the ABI version number (e.g., 0.7) matches. A change in API version may not result in a change in ABI version. In the 0.7.x series, the ABI version was always 0.7. Be aware, however, ABI versions on the v0.8 development branch have changed more frequently. For instance ABIs 0.8, 0.8-1 and 0.8-2 are not compatible.*
If building SoapySDR from source, **especially for use with a pre-packaged GNU Radio release, use the SoapySDR 'maint' branch.**" (caractères gras ajoutés pour expliciter le point important de la documentation) en nous expliquant que "*Ubuntu 22.04 and Fedora 36 both have ABI=v0.8. Nick says that the -2 is only on the development branch (which you and I both seem to use for our own builds).*" Tant que SoapySDR vit sa vie seul, la cohérence est garantie, mais tous les problèmes surviennent en tentant de faire cohabiter la bibliothèque propriétaire, la version compilée à la main de SoapySDR et les paquets de la distribution Debian/GNU Linux qui fournissent GNU Radio et `soapy*`. En effet, GNU Radio s'appuie sur SoapySDR, qui lui même va s'appuyer sur un service tournant en tâche de fond `sdrplay_apiService` qui lui même fait appel à la bibliothèque `libsdrplay_api.so`. En exécutant en l'état `2buildSoapy.sh`, les binaires incluent la version de l'ABI de la brache de développement (0.8-2 en date de rédaction) et toute incohérence se solde par un échec lors de l'interaction avec la Source Soapy de GNU Radio. Pour corriger, après avoir `git clone https://github.com/pothosware/SoapySDR` dans le script, ajouter l'utilisation d'une branche stable avec par exemple `git checkout maint`. Ainsi, l'identifiant mineur est retiré de la version de l'ABI pour se lier aux paquets `libgnuradio-soapy` de Debian. La situation actuelle du script ne semble pas produire de problème pour les utilisants d'Ubuntu qui semble générer ses paquets depuis la branche `master` du dépôt et s'accomode donc de l'identifiant mineur de l'ABI. Maintenant que `2buildSoapy.sh` a été complété avec (une seule ligne ajoutée par rapport au script existant)

```
git clone https://github.com/pothosware/SoapySDR
cd SoapySDR
git checkout maint
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr ..
make
sudo make install
```

nous nous assurons que la compilation se conclut avec

```
Set runtime path of "/usr/local/...SoapySDR/modules0.8/..."
```

dont l'absence d'extension mineure (`SoapySDR/modules0.8-2`) garantit la suite des opérations.

La liste des périphériques supportés par SoapySDR est accessible par

```
$ SoapySDRUtil --find
######################################################
##     Soapy SDR -- the SDR abstraction library     ##
######################################################

Found device 0
  driver = sdrplay
  label = SDRplay Dev0 RSP1 B0003P0001
  serial = B0003P0001

Found device 1
  driver = miri
  label = Mirics MSi2500 default (e.g. VTX3D card)
  miri = 0
```

dont les détails des conditions de fonctionnement sont fournies par

```
$ SoapySDRUtil --probe="driver=sdrplay"
######################################################
##     Soapy SDR -- the SDR abstraction library     ##
######################################################

----------------------------------------------------
-- Device identification
----------------------------------------------------
  driver=SDRplay
  hardware=RSP1
  sdrplay_api_api_version=3.070000
  sdrplay_api_hw_version=1

----------------------------------------------------
-- Peripheral summary
----------------------------------------------------
  Channels: 1 Rx, 0 Tx
  Timestamps: NO
  Other Settings:
     * RF Gain Select - RF Gain Select
       [key=rfgain_sel, default=1, type=string, options=(0, 1, 2, 3)]
     * IQ Correction - IQ Correction Control
       [key=iqcorr_ctrl, default=true, type=bool]
     * AGC Setpoint - AGC Setpoint (dBfs)
       [key=agc_setpoint, default=-30, type=int, range=[-60, 0]]

----------------------------------------------------
-- RX Channel 0
----------------------------------------------------
  Full-duplex: NO
  Supports AGC: YES
  Stream formats: CS16, CF32
  Native format: CS16 [full-scale=32767]
  Antennas: RX
  Corrections: DC removal
  Full gain range: [0, 42] dB
    IFGR gain range: [20, 59] dB
    RFGR gain range: [0, 3] dB
  Full freq range: [0.01, 2000] MHz
    RF freq range: [0.01, 2000] MHz
    CORR freq range:  MHz
  Sample rates: 0.0625, 0.096, 0.125, 0.192, 0.25, ..., 6, 7, 8, 9, 10 MSps
  Filter bandwidths: 0.2, 0.3, 0.6, 1.536, 5, 6, 7, 8 MHz
```

qui indique que le RSP1 supporté par les pilotes `miri` ou `sdrplay` est identifé. Les fonctionnalités du pilote libre `miri` sont trop rudimentaires pour être développées mais les performances annoncées de RSP1 avec 10 Méchantillons/s et une bande passante de 8 MHz sont tout à fait intéressantes. En effet, au contraire de cette souplesse d'emploi avec l'accès sans interruption au spectre de 0,01 à 2000 MHz et des fréquences d'échantillonnage de 0,2 à 8 MS/s fournies par le pilote `sdrplay`, nous constatons que le

pilote `miri` annonce des performances beaucoup plus restreintes avec un spectre discontinu et une unique fréquence d'échantillonnage imposée à 8 MS/s :

```
Probe device driver=miri
_mirisdr_alloc_async_buffers

------------------------------------------------
-- Device identification
------------------------------------------------
  driver=miri
  hardware=miri

------------------------------------------------
-- Peripheral summary
------------------------------------------------
  Channels: 1 Rx, 0 Tx
  Timestamps: NO

------------------------------------------------
-- RX Channel 0
------------------------------------------------
  Full-duplex: NO
  Supports AGC: NO
  Stream formats: CF32
  Native format: CF32 [full-scale=1]
  Antennas: RX
  Full gain range: [-1, 49, 1] dB
    LNA gain range: [-1, 49, 1] dB
  Full freq range: [0.15, 30], [64, 108], [162, 240], [470, 960], [1450, 1675] MHz
    RF freq range: [0.15, 30], [64, 108], [162, 240], [470, 960], [1450, 1675] MHz
    CORR freq range:  MHz
  Tune args:
     * LO Offset - Tune the LO with an offset and compensate with the baseband CORDIC.
       [key=OFFSET, units=Hz, default=0.0, type=float]
     * CORR - Specify a specific value for this component or IGNORE to skip tuning it.
       [key=CORR, units=Hz, default=DEFAULT, type=float, options=(DEFAULT, IGNORE)]
  Sample rates: 8 MSps
```

Nous pouvons finalement générer la première chaîne de traitement pour acquérir et traiter le flux de données depuis GNU Radio (Fig. 4), par exemple pour écouter simultanément toutes les stations d'une bonne partie de la bande FM [5].

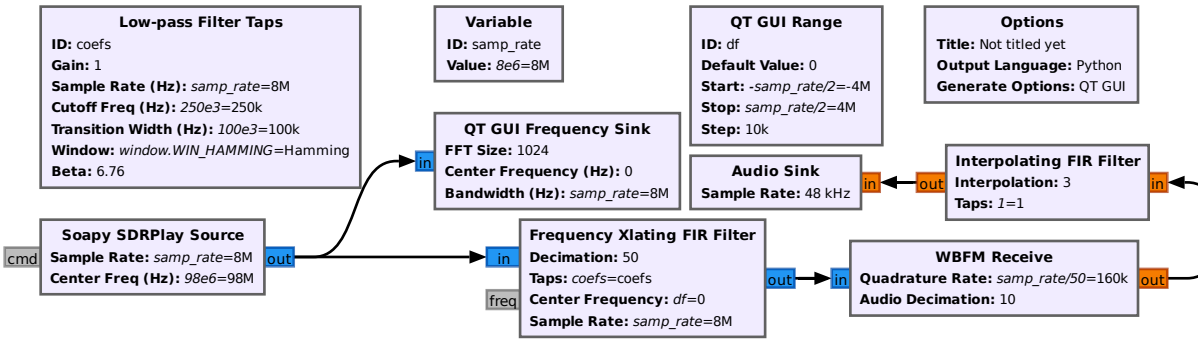

FIGURE 4 – Chaîne de traitement de la bande FM dans laquelle le récepteur RSP1 centré sur 98 MHz acquiert 8 MHz : balayer la fréquence du `Xlating FIR Filter` permet de démoduler chaque station dans la bande passante, et donc potentiellement de démoduler toutes ces stations FM simultanément si la puissance de calcul le permet, et ce sans jamais changer la fréquence de l'oscillateur local mais uniquement en transposant numériquement les bande allouées à chaque station FM vers la bande de base.

À l'exécution, ce script doit indiquer la détection de RSP1 par

```
[INFO] devIdx: 0
```

```
[INFO] SerNo: B0003P0002
[INFO] hwVer: 1
[INFO] rspDuoMode: 0
[INFO] tuner: 1
[INFO] rspDuoSampleFreq: 0,000000
[INFO] Using format CF32.
```

et le résultat à Besançon est illustré en Fig. 5 avec la multitude de canaux de la bande FM accessible simultanément, sans atteindre les 18 chaînes reçues simultanément dans la bande de 8 MHz à Paris. Noter qu'alors nous avions pris l'habitude d'ajuster la fréquence d'échantillonnage radiofréquence des RTL-SDR à un multiple de 48 kHz comprise entre 1 et 2 MHz pour atteindre la fréquence d'échantillonnage audiofréquence de 48 kHz par décimations entières lors du filtre passe-bas de sélection de la station FM puis lors de la démodulation FM, le peu de fréquences d'échantillonnage accessibles au RSP1 imposent d'interpoler pour atteindre la fréquence audio visée, dans cet exemple $8 \cdot 10^6/10/5 \times 3 = 48000$.

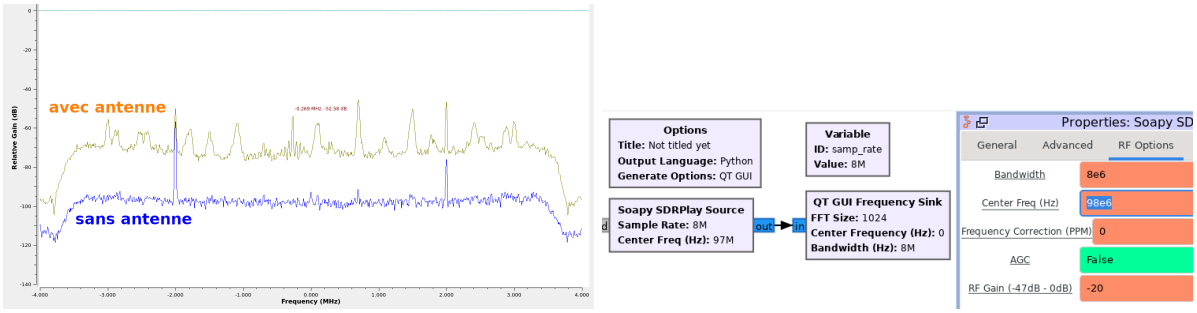

FIGURE 5 – Gauche : mesure de la bande FM à Besançon sur une largeur de 8 MHz centrée sur 98 MHz, permettant de recevoir simultanément 12 stations. Droite : réglages du bloc Soapy SDR Play en prenant soin de préciser la bande passante de mesure `Bandwidth`. Noter les deux parasites à ±2 MHz, peu élégants mais aisément filtrés numériquement et qui n'handicaperont pas la démodulation FM large bande.

En l'absence de RSP1 branchée, GNU Radio fournira le message quelque peu cryptique de "`RuntimeError: no available RSP found`" qui n'est pas beaucoup plus limpide que l'incohérence d'ABI qui se traduit par "`SoapySDR::DEvice::make() no match`".

À l'issue de la modification matérielle du récepteur pour ajouter le connecteur sur les broches de réception de la bande L, nous testons la capacité de réception par GNU Radio autour de 1200 MHz en injectant un signal d'un synthétiseur de fréquence émettant −80 dBm pour ne pas risquer d'endommager l'étage de réception du MSi001. Nous constatons sur la Fig. 6 que le signal est parfaitement observé, malgré nombre de parasites qui sont visibles sur le spectre (droite) mais qui ne dérangeront pas le traitement de radio logicielle qui va nous intéresser. De telles parasites étaient déjà visibles dans la bande FM et nous avons vérifié qu'ils sont bien produits par le circuit en les observant même lors d'une acquisition dans une cage de Faraday. On notera que le synthétiseur TTi TGR2053 utilisé ici cesse de fonctionner lorsque la température dépasse les 35 degrés, cause de bien des déboires lors de nos tests au cours de l'été !

La détection du récepteur par SoapySDR et son utilisation par le bloc `Soapy SDRPlay Source` de GNU Radio 3.10.2 sont mis en évidence par un cercle vert sur la Fig. 6. Le récepteur a volontairement été programmé avec un écart de 1 MHz par rapport à la fréquence de la source pour ne pas se placer sur le pic DC de la fuite potentielle de l'oscillateur local, astuce classique pour s'affranchir des bruits basse fréquence du détecteur en se décalant de la porteuse visée et en finalisant l'acquisition par une transposition numérique vers la bande de base par le `Xlating FIR Filter`, version numérique du super-hétérodynage inventé il y a un siècle pour pallier les déficiences des tubes à vides d'antan.

4 Réception des signaux issus d'un satellite géostationnaire

Maintenant que nous sommes convaincus de pouvoir recevoir un signal au-dessus du GHz et avec plus de 5 MHz de bande passante tel qu'imposé par le canal de communication avec le satellite géostationnaire,

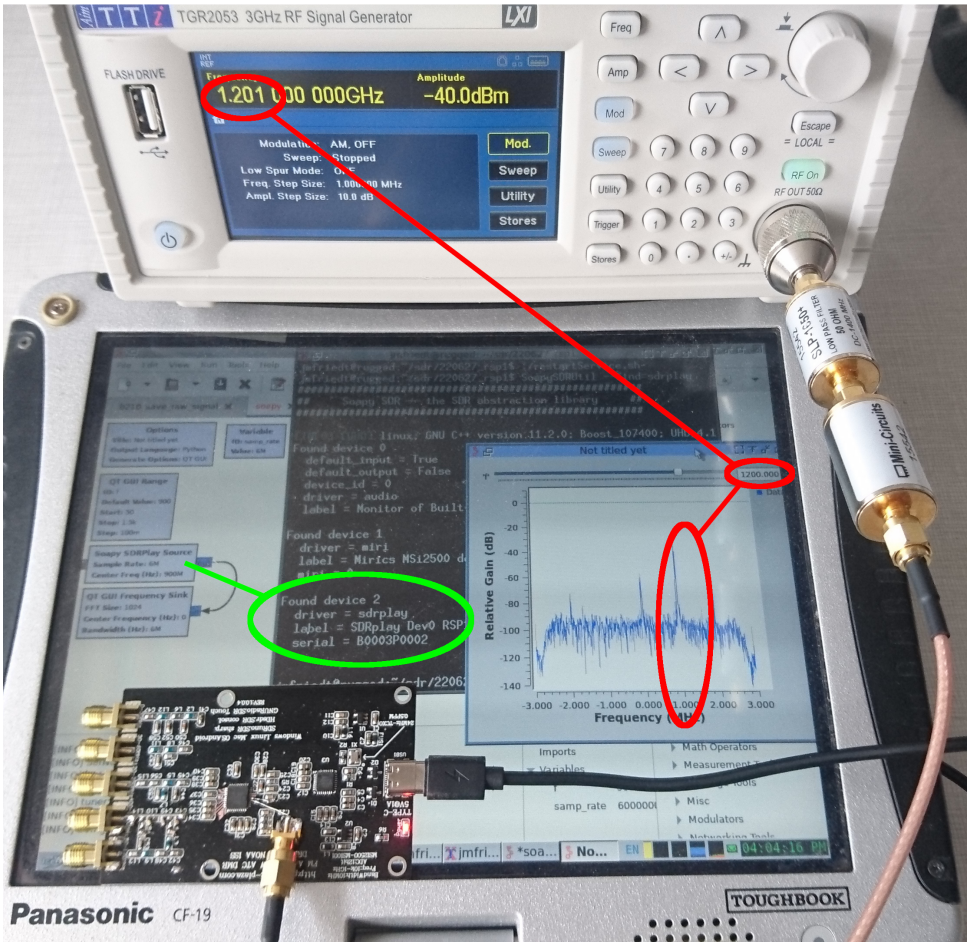

FIGURE 6 – Validation de la capacité du RSP1 modifié par l'ajout du connecteur sur l'entrée de la bande L à recevoir un signal autour de 1200 MHz (ellipses rouges) lorsque reglé sur cette porteuse (ellipses vertes).

pourrons nous décoder un signal de transfert de temps et fréquence échangé au travers du satellite géostationnaire Telstar-11N ?

Muni d'un T de polarisation pour alimenter le récepteur (*Low Noise Block* – LNB) au foyer de la parabole des 13 V nécessaires au fonctionnement de son oscillateur local et ses amplificateurs (Fig. 7), le RSP1 enregistre quelques secondes du signal en heure paire (transmission active) et heure impaire (transmission inactive) pour vérifier que le signal de transfert de temps et fréquence échangé par les laboratoires de métrologie européens n'est présent que dans le premier cas (Fig. 8). La mise au carré du signal modulé en BPSK annule l'étalement de spectre introduit par la modulation de phase et accumule l'énergie dans une porteuse pure au double de l'écart de fréquence entre oscillateur local du satellite et oscillateur local du LNB au foyer de la parabole ou oscillateur local du RSP1. Ces raies spectrales sont parfaitement visibles sur le graphique du haut de Fig. 8. Afin de se convaincre qu'il ne s'agit pas de raies parasites, nous corrélons le signal centré exactement sur 0 Hz après identification du double écart de fréquence $2\delta f$ par transformée de Fourier du carré du signal et compensation numérique par multiplication avec un oscillateur numérique de la forme $\exp(2\pi\delta f \times t)$ avec $t = [0 : N]/fs$, N le nombre d'échantillons IQ acquis et fs la fréquence d'échantillonnage, avec le code pseudo-aléatoire de chaque station émettrice.

La présence des pics de corrélation (Fig. 9) toutes les 4 ms ou 32000 échantillons acquis au rythme de 8 MS/s, avec des retards différents pour chaque observatoire distribué en Europe, prouve que le signal reçu est le bon et possède bien toutes les caractéristiques attendues du signal diffusé par Telstar-11N vers l'Europe au cours des heures paires.

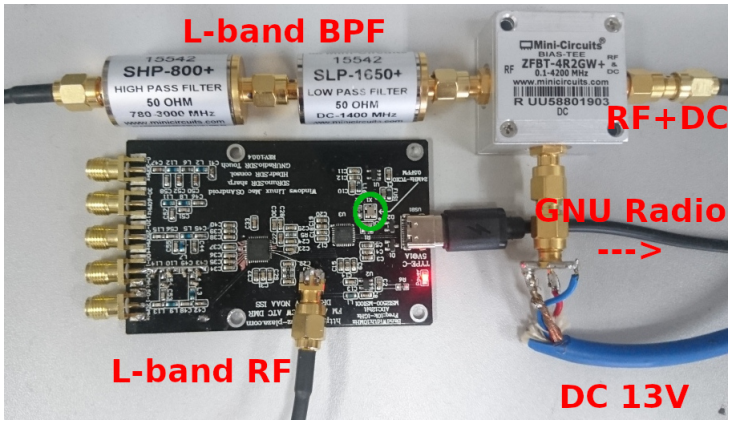

FIGURE 7 – Montage expérimental incluant un filtre passe-bande (formé d'un filtre passe-haut et un filtre passe-bas en série) et un T de polarisation pour alimenter le LNB au foyer de la parabole pointée sur Telstar-11N. Le cercle vert met en évidence l'oscillateur local à 24 MHz de plutôt bonne exactitude face à ce que les récepteurs RTL-SDR nous avaient habitués.

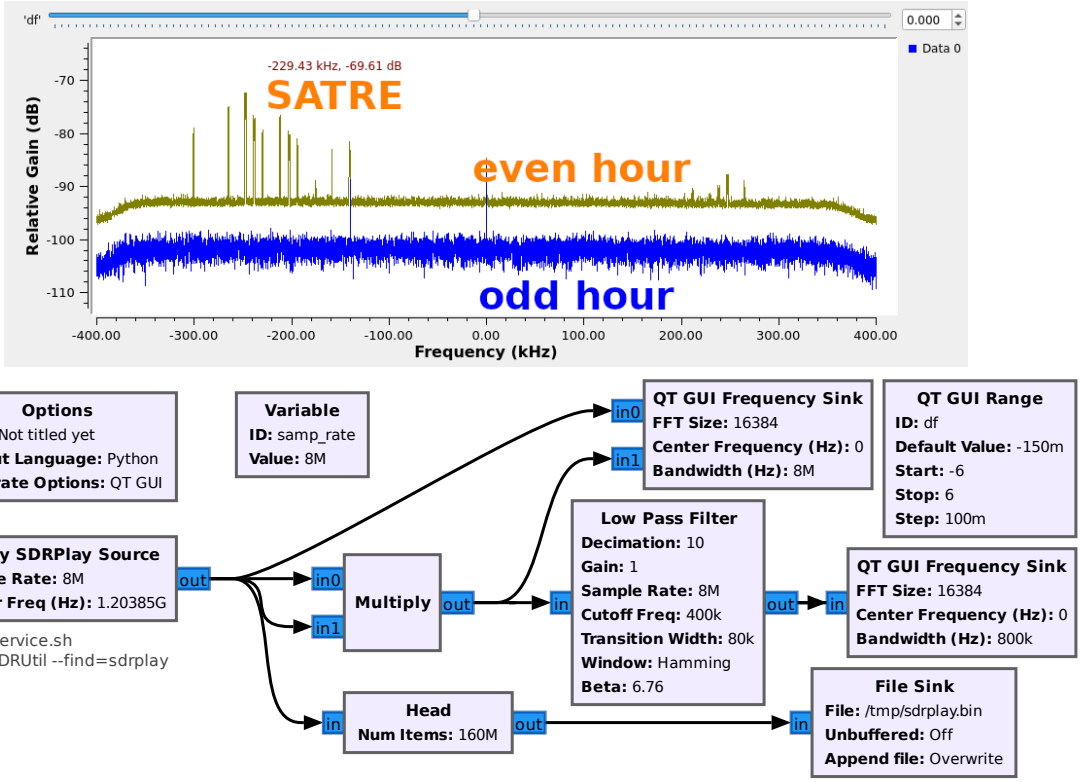

FIGURE 8 – Haut : spectre des signaux reçus autour de 1,204 GHz après transposition de fréquence par le LNB de la parabole pointée vers Telstar-11N en heure paire lorsque les signaux de synchronisation de temps et fréquence sont transmis (vert) et en heure impaire de silence des liaisons radiofréquences (bleu). Bas : la chaîne d'acquisition avec mise au carré du signal pour visualiser les porteuses accumulant toute l'énergie lors de l'annulation de la modulation BPSK.

Grâce à ce montage, il est envisageable de caractériser, voir de corriger, la dérive en fréquence de l'oscillateur local. Pour ce faire, nous pourrions remplacer l'oscillateur local fixe à 24 MHz (cercle vert sur Fig. 7) par une source programmable de fréquence – par exemple un synthétiseur numérique de

fréquence (DDS) tel que l'AD9832 – et programmer la fréquence en observant le décalage à la valeur nominale de l'écart de temps entre deux pics de corrélation. Sachant que l'incertitude sur la position du pic de corrélation, après application d'un ajustement parabolique pour améliorer la précision sur le délai par rapport à la période d'échantillonnage égale au rapport signal à bruit de la mesure – est d'une fraction de nanosecondes toutes les 4 ms ou la centaine de picosecondes toutes les minutes, nous devrions pouvoir stabiliser la source de fréquence à une stabilité relative de quelques 10^{-10}. Le problème de l'exactitude en temps est décrit dans l'article dédié à ce sujet [1] et est plus complexe compte tenu de la variation de la position du satellite autour de sa position d'équilibre.

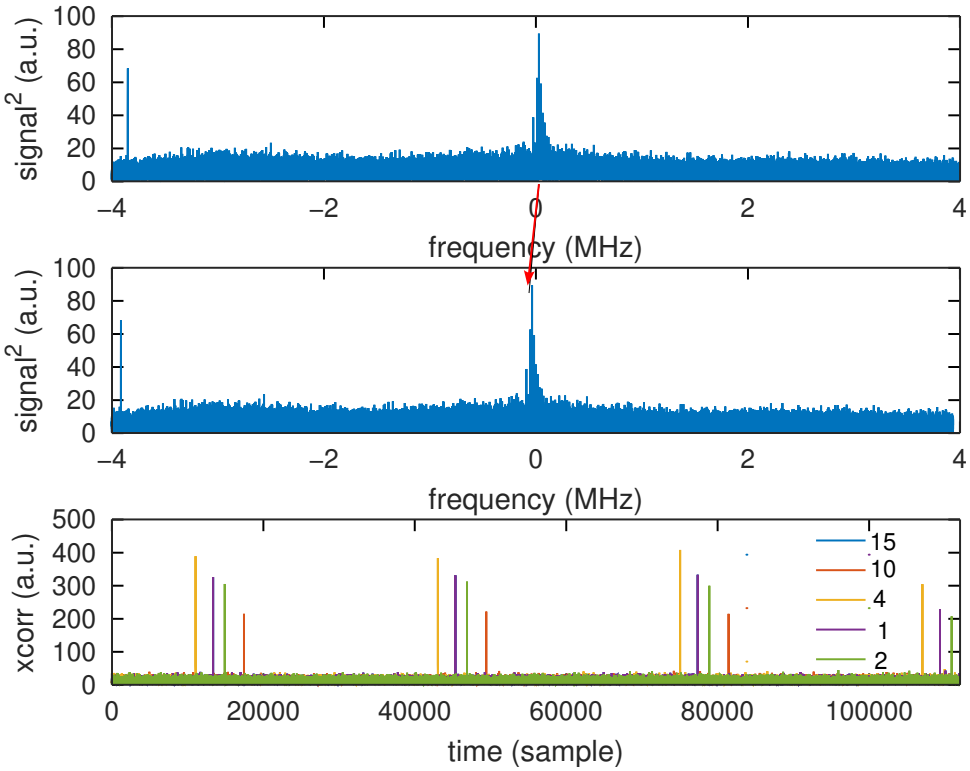

FIGURE 9 – Démonstration de la cohérence des signaux acquis par détection des pics de corrélation toutes les 4 ms ou 32000 échantillons acquis au rythme de 8 MS/s (bas) après avoir ramené les signaux en bande de base à 0 Hz exactement par transposition numérique de fréquence (haut → milieu).

5 Conclusion

Nous avons exploré l'utilisation du RSP1, clone chinois de SDRPlay du même nom, disponible pour une vingtaine d'euros. Alors qu'il peut sembler discutable de promouvoir un clone de produit développé en Angleterre, ce circuit ne fait qu'assembler deux composants conçus pour être appairés – MSi001 et MSi2500 – et il est tout aussi discutable de voir le concepteur européen commercialiser à près de 100 euros un produit fabriqué pour un coût qui ne peut dépasser le prix de vente du clone constaté actuellement. Ce faisant, nous avons découvert une alternative crédible au récepteurs RTL-SDR, au prix d'une bibliothèque propriétaire fort peu agréable à installer mais qui finit par fournir le service attendu. La pérénité de la solution n'est pas garantie et seule la transition vers la bibliothèque libre garantira une solution adaptable à toute architecture de processeur/évolution de GNU/Linux.

Ce montage, couplé à une parabole de réception de signaux de télévision transmis par satellite, a permis de recevoir les signaux de transfert de temps et de fréquence depuis un satellite géostationnaire avec un rapport signal à bruit pour en décoder le contenu, avec la perspective d'asservir l'oscillateur local sur ces signaux de référence issus d'horloges atomiques participant au Temps Atomique International.

Références

[1] J.-M Friedt *Le temps et son transfert par satellite géostationnaire : réception avec une parabole de télévision et d'une radio logicielle*, Hackable **44** (2022)

[2] un certain nombre de notes plus ou moins exactes (tel qu'annoncé !) mais informatives pour aborder le RSP1 à `https://github.com/EndlessEden/msiSDR/tree/RSP1-S`

[3] `https://www.kechuang.org/t/83757` s'inspire de SDRPlay qui est cité sur la page web décrivant ce projet pour rerouter complètement un circuit équivalent.

[4] *About SDRplay - the history behind the RSP* (2016) à `https://www.youtube.com/watch?v=Sl1O5gf3AQE`

[5] B. Happi Tietche, *Proposition d'architectures radio logicielles FPGA pour démoduler simultanément et intégralement les bandes radios commerciales, en vue d'une indexation audio*, thèse de l'Université Pierre et Marie Curie (2014) à `https://www.theses.fr/2014PA066052.pdf`