Buildroot pour la compilation croisée de GNU Radio pour plateformes embarquées

La Raspberry Pi {3,4} (RPi) est un ordinateur architecturé (pour les caractéristiques nous intéressant en radio logicielle) autour d'un processeur quad-cœurs ARM cadencés à 1,5 GHz avec 1,2 ou 4 GB de mémoire volatile. Le système d'exploitation est stocké sur carte uSD et sera donc simple à mettre à jour depuis un ordinateur (ne *jamais* compiler sur la cible embarquée). La génération d'une chaîne de compilation dédiée – contrairement à l'utilisation d'une distribution binaire – permet d'optimiser le jeu d'instructions pour le processeur spécifique à la cible considérée.

Notre objectif va être d'**exécuter GNU Radio sur RPi4** en vue d'effectuer un certain nombre de traitements sur la plateforme embarquée avant d'en envoyer le résultat vers le PC (exemple : recevoir une station FM sur la RPi et envoyer par une communication Zero-MQ le signal audio vers le PC).

1 Buildroot pour RPi

Buildroot est un environnement fournissant un ensemble cohérent de
- chaîne de compilation croisée sur l'hôte (*host* Intel x86)
- bibliothèques et applications en espace utilisateur sur la cible (*target* ARM)
- noyau Linux sur la cible
- *bootloader* (séquence de démarrage) pour initialiser le processeur et ses périphériques sur la cible en vue de charger le noyau Linux qui supervisera les applications en espace utilisateur.

Cette cohérence évite bien des déboires lors de la compilation sur l'hôte d'applications ou module noyau pour la cible (on ne compile **jamais** sur la cible de ressources réduites et dont la vocation est de contrôler un système embarqué et non d'être muni d'un compilateur aussi lourd et complexe que `gcc`).

Pour installer Buildroot sur Raspberry Pi 4, tel que décrit à `https://github.com/buildroot/buildroot/tree/master/board/raspberrypi` mais la documentation de cette page n'est pas à jour (?!) :

1. `git clone https://github.com/buildroot/buildroot`
2. `cd buildroot`
3. `make raspberrypi4_64_defconfig`

qui récupère l'arborescence de Buildroot, et la configure pour la RaspberryPi4 en mode 64 bits (pour la RPi3, sélectionner `raspberrypi3_64_defconfig`). Puisque le support Python de GNU Radio nécessitera la bibliothèque `glibc` au lieu de `uClibc` sélectionnée par défaut, nous ajustons la configuration initiale par

4. `make menuconfig`
5. Toolchain → C library (uClibc-ng) → glibc
6. Exit

Une fois la bonne bibliothèque sélectionnée

7. `make`

compile l'ensemble des outils. Cette opération prend environ 40 minutes sur un processeur Xeon à 8 cœurs cadencés à 2,33 GHz avec une connexion internet rapide, et occupe environ 7.4 GB sur le disque dur. Au cours de la compilation, Buildroot n'affecte *que les fichiers contenus dans le répertoire* `output`. Ainsi, effacer ce répertoire (et ses sous-répertoires) permet de repartir d'une version propre de Buildroot. Dans `output`, tout ce qui est relatif à l'hôte x86 se trouve dans `host`, tout ce qui est relatif à la cible dans `target`. Nous n'aurons pas ici besoin de regarder le contenu du répertoire contenant les sources des outils compilés mais serons susceptibles d'en effacer certains contenus pour en forcer la recompilation dans `build`. Enfin pour ce qui nous intéresse maintenant, le résultat de la compilation se trouve dans `images`.

Dans ce répertoire `output/images`, nous trouvons `sdcard.img` qui est l'image à transférer sur la carte uSD en vue d'en exécuter le contenu sur RPi. Ici, "transférer" ne signifie par copier car nous devons écrire octet par octet le contenu du fichier .img sur la carte. Cette opération est prise en charge sous GNU/Linux par `dd`.

⚠ La ligne qui va suivre peut **corrompre le disque dur** si le mauvais périphérique est sélectionné. Toujours **vérifier** le nom du périphérique associé à la carte SD (`dmesg | tail`) avant de lancer la commande `dd`.

L'image résultant de la compilation est transférée sur la carte SD par

sudo dd if=output/images/sdcard.img of=/dev/sdc

où nous avons volontairement choisi le périphérique `/dev/sdc` dans cet exemple car rarement utilisé : en pratique, la carte SD sera souvent nommée `/dev/sdb` (second disque dur compatible avec le pilote SCSI de Linux) ou `/dev/mmcblk0` (cas du lecteur SD interne).

En cas d'utilisation d'un gestionnaire de fichiers ou de bureau, bien vérifier que la carte SD n'est pas prise en charge par ces outils (*eject*) qui risquent d'interférer avec `dd`.

⚠ **Attention** : le contenu de la carte SD, ou de tout support pointé par le dernier argument de cette commande, sera **irrémédiablement** perdu. Vérifier à deux fois le nom du périphérique cible de l'image issue de buildroot.

Une fois l'image flashée sur la carte SD, nous constaterons deux partitions : une première en VFAT (format compatible Microsoft Windows) avec le devicetree, le noyau Linux et le bootloader, et une seconde partition contenant le système GNU/Linux (`rootfs`)

2 Ajouter des packages : BR2_EXTERNAL

Pour le moment nous avons compilé une image standard Buildroot sans support de GNU Radio. Ces paquets dédiés ne sont pas sélectionnés par défaut mais peuvent être activés. Pour ce faire, exécuter dans le répertoire de Buildroot la commande `make menuconfig` et sélectionner `Target packages`. La recherche ("/" comme dans `vi`) permet de facilement trouver l'emplacement d'un paquet, par exemple GNU Radio.

Jusqu'ici nous n'avons travaillé qu'avec les paquets Buildroot "officiels" maintenus par la communauté des développeurs de Buildroot. Certains paquets ne sont pas encore intégrés sur le site officiel mais peuvent néanmoins compléter l'installation en cours grâce au mécanisme de `BR2_EXTERNAL`. Un exemple est le support de la PlutoSDR grâce à `gr-iio`, qui est fourni dans le dépôt `BR2_EXTERNAL` disponible à `https://github.com/oscimp/PlutoSDR` et plus spécifiquement dans la branche `for_next`. Ainsi, après être sorti du répertoire Buildroot pour créer une nouvelle arborescence :

1. `git clone https://github.com/oscimp/PlutoSDR`
2. `cd PlutoSDR`
3. `git checkout for_next`
4. `source sourceme.ggm`

Maintenant que le dépôt `BR2_EXTERNAL` a été téléchargé, la branche appropriée sélectionnée, et les variables d'environnement définies (dernière commande), retourner dans le répertoire de Buildroot et `make menuconfig`. L'exécution de `make menuconfig` donne maintenant accès à un nouveau menu `External options` qui inclut `gr-iio`, `libuhd` ou `gnss-sdr`.

1. `make menuconfig`
2. `/eudev`
3. Sélectionner la dernière option indiquée par `BR2_ROOTFS_DEVICE_CREATION_DYNAMIC_EUDEV` et remplacer `/dev management` par `Dynamic using devtmpfs + eudev`
4. `/python3`
5. Sélectionner l'option (4) indiquée par `BR2_PACKAGE_PYTHON3`
6. `/gnuradio`
7. Sélectionner l'option (1) indiquée par `BR2_PACKAGE_GNURADIO`
8. Sélectionner les options additionnelles de GNU Radio selon les besoins (nous aurons besoin de `gr-zeromq support` et `python support`)
9. `/osmosdr`

10. Sélectionner `BR2_PACKAGE_GR_OSMOSDR` (avec support Python et support Osmocom RTLSDR)
11. dans `External options` sélectionner `uhd` et pour la B210 `b200 support` et `python API support`,
12. dans `External options` sélectionner `gr-iio` si la PlutoSDR est utilisée.

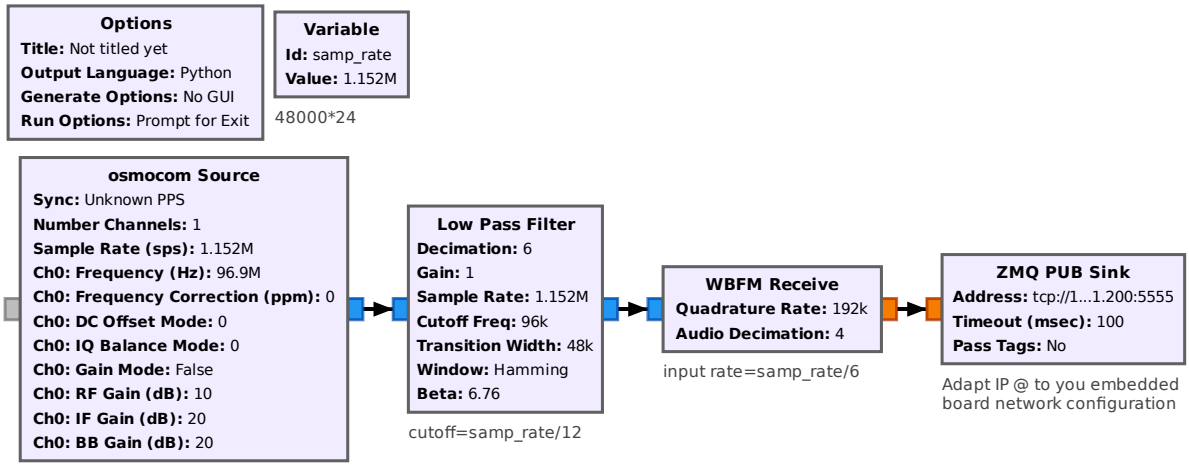
Le fichier résultant fera environ 550 MB, nécessitant d'augmenter la taille disponible dans `.config` par `BR2_TARGET_ROOTFS_EXT2_SIZE="420M"`.

Nous pouvons ajuster la configuration avant de `dd` l'image sur la carte SD en ajoutant des fichiers dans `output/target`, par exemple une configuration statique du réseau dans `etc/network/interfaces`, ou copier les *firmware* des USRPs depuis le PC hôte dans le sous répertoire `usr/share/uhd/images` de `output/target` afin que ces fichiers soient ultérieurement disponibles sur la cible embarquée. Une fois le contenu de `output/target` convenablement ajusté, retourner dans le répertoire racine de Buildroot et exécuter `make` pour régénérer le fichier `output/images/sdcard.img`.

3 GNU Radio sur RPi

À titre d'illustration de la façon que nous abordons d'utiliser GNU Radio sur plateforme embarquée, nous générons en utilisant GNU Radio Companion sur le PC une application en ligne de commande ("No GUI") puisque évidemment aucune interface graphique ne devrait être disponible sur la cible embarquée. Le script Python3 résultant sera exécuté sur la RPi. Le flux audio-fréquence résultant de la démodulation du signal de la bande FM commerciale sera transmis au PC pour être joué sur la carte son.

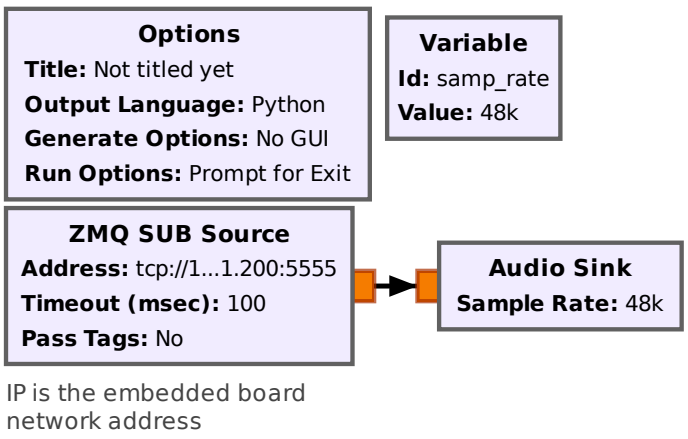
Sur le PC, lancer GNU Radio Companion (fourni par GNU Radio 3.8) et générer la chaîne de traitement suivante :

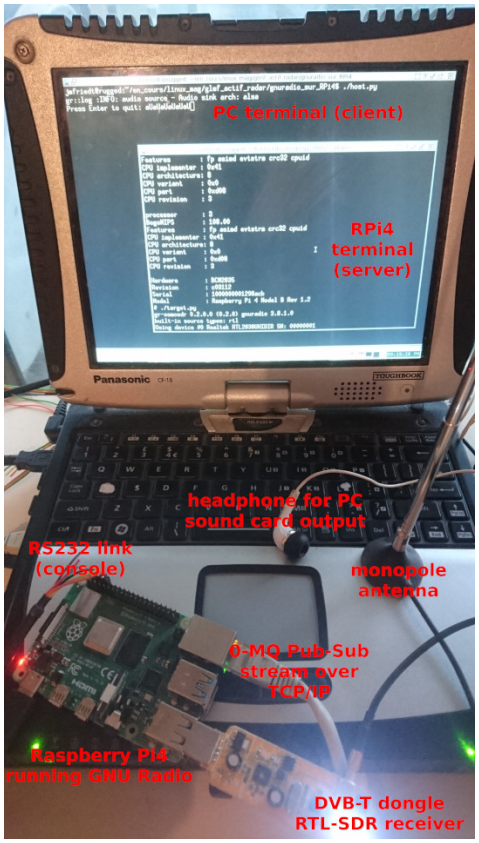


Le script Python ainsi généré est transféré à la RPi. Bien prendre soin d'adapter l'adresse IP de la liaison TCP sur 0-MQ à l'adresse de la RPi : le serveur est exécuté sur la plateforme embarquée et en utilisant une liaison de type Publish-Subscribe (s'apparentant à une liaison UDP), tout client se connectant au serveur exécuté sur la cible embarquée peut recevoir le flux de données. L'adresse IP se incluse idéalement dans le sous-réseau du PC pour simplifier la configuration du routage, tandis que le port peut être toute valeur au dessus de 1024. La seule contrainte sur cette chaîne de traitement est d'aboutir à la fin à une fréquence d'échantillonnage égale à une valeur compatible avec la cartes son du PC (ici 48 kHz) après une séquence de décimations par des facteurs entiers sélectionnée ici par le choix d'une fréquence d'échantillonnage initiale de 48 kS/s. Le premier filtre passe-bas sélectionne une unique station FM tout en gardant assez de bande passante ($\geq$200 kHz) pour la démodulation FM à bande large, et le démodulateur FM ajoute un second étage de décimation.

4 Communication RPi vers PC

Après traitement des données radiofréquences brutes (I/Q) acquises sur la bande FM commerciale par la RPi, et avoir pré-traité le signal FM sur la plateforme embarquée, le flux audio-fréquence est transmis vers le PC par 0-MQ.





Gauche : chaîne de traitement du client, qui récupère un flux au format *subscribe* de 0-MQ et alimente la carte son du PC. Droite : montage expérimental, avec une RPi4 connectée par un port série virtuel et par Ethernet au PC portable. La RPi4 échantillonne le flux de coefficients I/Q du récepteur de télévision numérique terrestre utilisé comme source de radio logicielle en ajustant sa fréquence dans la bande FM, et transmet le flux audio-fréquence après démodulation au PC, permettant d'écouter le programme grâce à un casque connecté à la sortie de la carte son. Cette figure ne permet pas d'illustrer l'excellente qualité audio entendue à la sortie de la carte son, démontrant le bon fonctionnement de ce montage.

5 Développements logiciels

Nous nous intéressons à modifier les fonctionnalités de `gnss-sdr`. Le code source de ce logiciel a été téléchargé et placé dans le répertoire `output/build` lors de sa sélection et installation. Le répertoire de compilation pour la cible se trouve à `output/build/gnss-sdr-0.0.12/buildroot-build/` tandis qu'un répertoire séparé `output/build/gnss-sdr-0.0.12/build` permet de simultanément tester les modifications aux codes sources sur le PC hôte. Le résultat de la compilation (`make`), soit dans `buildroot-build` (cible ARM) ou `build` (cible x86), se trouve dans `src/main/gnss-sdr`.