Radio logicielle 4/4 : GNU Radio “avancé” pour communiquer avec le monde extérieur, et blocs de traitements dédiés

J.-M Friedt

FEMTO-ST/département temps-fréquence, Besançon

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

15 janvier 2025

GNU Radio v.s ...

- GNU Radio traite efficacement un flux *continu* de données I+jQ...
- mais s'accomode mal de paquets (PDU, *Protocol Data Unit*) des couches plus élevées OSI

$\Rightarrow$ partage des tâches entre outils appropriés.

GNU Radio doit donc communiquer avec le monde extérieur

- *pipe* nommé (`mknod` sous Unix), symbole | du shell
- communication TCP/IP ou UDP/IP et associés
- contrôle de la chaîne de traitement

Fonctionnalité de traitement dédiées (Python)

Contrôle du flux de données

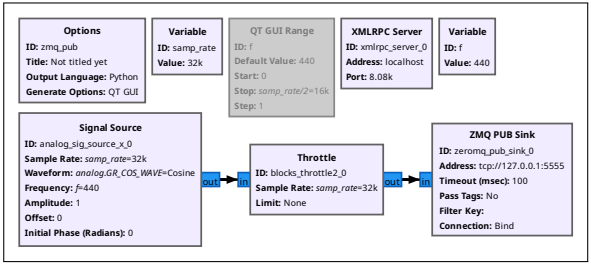
- GNU Radio Companion est un générateur de code Python faisant appel aux blocs de traitement Python ou C++,
- ... mais si le code Python est modifié, impossible de revenir en arrière.
- `Python Module` : programme Python inclus (`import ...`) dans le programme
- `Python Snippet` : un bout de code appelé en divers points d'exécution du programme

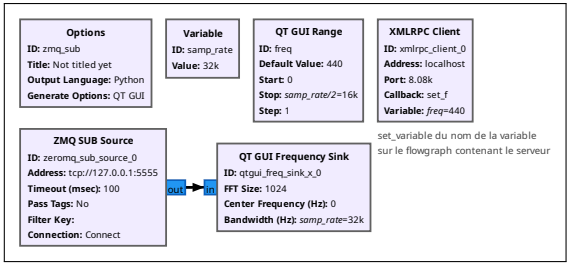
Ces codes Python n'ont **pas accès** aux flux $I + jQ$ mais ne peuvent que paramétrer la chaîne de traitement (fonctions de callback associées à chaque variable)

- Un *thread* séparé du programme principal peut gérer la communication ou des évènements périodiques

Standardisation des appels : XMLRPC

XMLRPC [1] : encapsulation standardisée (XML) des *Remote Procedure Call* (RPC)





Python :

```
from xmlrpc.client
import ServerProxy
s=ServerProxy('http://localhost:8080')
s.set_f(5000)
```

Bash :

```
xmlrpc localhost:8080 set_f i/1664
```

Chaîne XML transmise par POST :

```
curl -X POST -H 'Content-Type: text/xml' -d '<methodCall>\n<methodName>set_f</methodName>\
\n<params><param><value><i4>1664</i4></value></param></params>\n</methodCall>'\
'http://localhost:8080/RPC2'
```

1. https://gitlab.com/gnuradio_book/flowcharts/-/tree/main/chap3_communication

Python Block pour traiter le flux de données

- Le bloc possède un constructeur `__init()__` (initialisation des variable)
- ... qui définit la liste des paramètres (arguments fournis dans GNU Radio Companion) et entrées + sorties

```
def __init__(self, parametre1=1.0):
  gr.sync_block.__init__(
    self, name='Nom', in_sig=[], out_sig=[np.float32]
```

- une méthode `work()` est appelée continuellement par l'ordonnanceur en fournissant un vecteur de vecteurs en entrée

```
def work(self, input_items, output_items):
```

et produisant `len(output_items[0])` éléments en sortie

```
...
    return len(output_items[0])
```

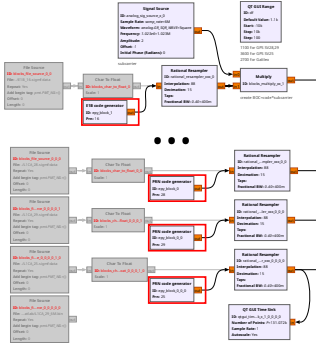
- Attention aux ressources de calcul ("L") lors du traitement en Python + console très lente de GNU Radio Companion ⇒ lancer le script Python depuis un terminal en cas d'affichages nombreux

Python Block pour générer les séquences GPS/Galileo

- Marc Lichtman (IQEngine) : rather than storing PRN sequence, generate on the fly.
- Python PRN generators found at `https://github.com/pmonta/GNSS-DSP-tools.git` in `gnsstools/gps` and `gnsstools/galileo`
- Copy-paste in GNU Radio `Python Block` with `in_sig=[], out_sig=[np.float32]`
- The scheduler requested number of items is unknown ⇒ fill with `intval` integer copies of the PRN and then circular buffer with the PRN sequence : `self.x=self.x[fracval:]+self.x[:fracval]` the fractional part of the length of the code returned in the output buffer

```
...
def e1b_code(prn):
  if prn not in codes:
    codes[prn] = e1b_parse_hex(prn)
  return codes[prn]
boc11 = np.array([1.0,-1.0])
...
class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
    def __init__(self, PRN=16):  # only default arguments here
        gr.sync_block.__init__(
            self, name='E1B code generator',
            in_sig=[], out_sig=[np.float32] # no input, only output
        )
        self.PRN = PRN
        self.x=list(1.-2.*e1b_code(self.PRN))
        self.xpos=0

    def work(self, input_items, output_items):
        intval=len(self.x)*(len(output_items[0])//len(self.x))  # integer number of copies of PRN →
            ↪ code
        fracval=len(output_items[0])-intval                      # fractional length of the PRN code
        output_items[0][0:intval] = self.x*(len(output_items[0])//len(self.x)) # *list = copy
        output_items[0][intval:intval+fracval] = self.x[0:fracval]
        self.x=self.x[fracval:]+self.x[:fracval]                 # rotate x
        return len(output_items[0])
```



Bloc de traitement en C++

- Même structure de Python : un constructeur, un destructeur et une fonction `work()`
- Génération de l'arborescence d'un bloc extérieur de traitement (`Out of Tree` – OOT [2]) [3])

  `$ gr_modtool newmod monmodule`

  pour créer le module `gr-monmodule`
- créer le bloc de traitement

  `$ cd gr-monmodule`

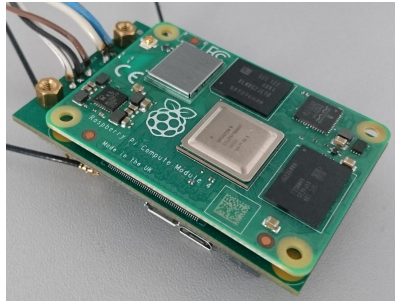
  `$ gr_modtool add monbloc`
- les fonctions se trouvent dans `lib/`, la description de l'interface graphique dans `grc` (format YAML), modifier `include/` pour ajouter des arguments en entrée
- si modification du contenu de `include`, `gr_modtool bind monbloc` pour re-générer les fichiers reliant Python et C++ (PyBIND [4])
- Complilation :

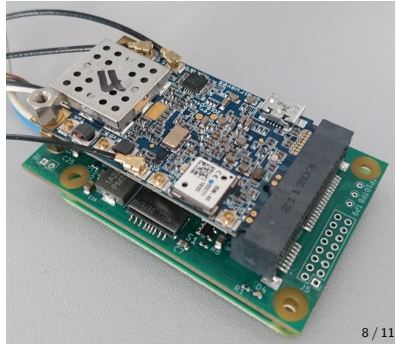
  ```
  $ mkdir -p build
  $ cd build
  $ cmake ../ && make
  $ sudo make insall # place monbloc dans /usr/local/
  ```

2. https://wiki.gnuradio.org/index.php?title=Creating_Python_OOT_with_gr-modtool
3. https://wiki.gnuradio.org/index.php?title=Creating_C%2B%2B_OOT_with_gr-modtool
4. voir https://github.com/jmfriedt/gr-m17 pour quelques rappels

GNU Radio sur système embarqué, ressources de calcul

- VOLK [a] et instructions SIMD (amd64 v.s ARM)
- GNU Radio dans Buildroot [b]
- Pour le moment, seul GPS L1 C/A peut être traité en temps-réel
- Exemple : LimeSDR XTRX [c] $50,8\ \text{mm} \times 20,97\ \text{mm}$ sur ComputeModule4 (Raspberry Pi 4 OEM)





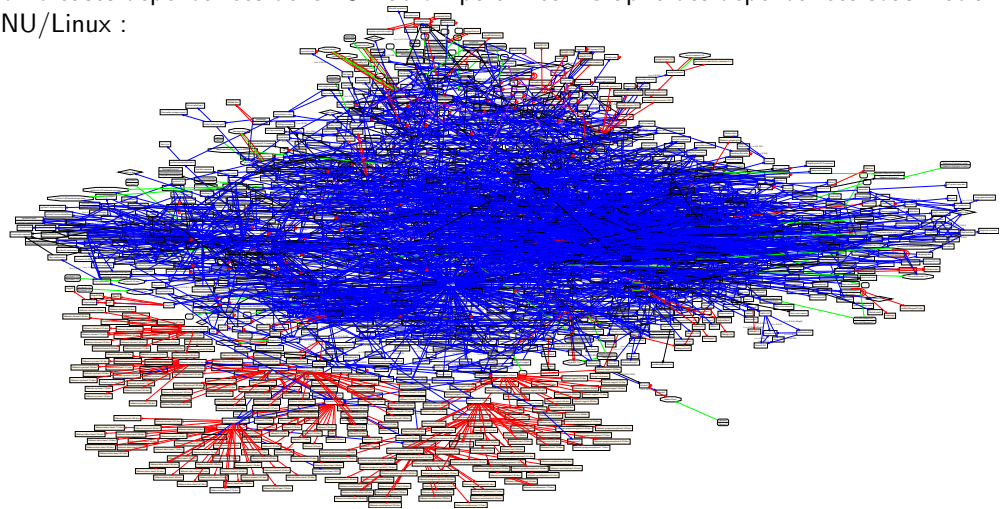
a. *Vector-Optimized Library of Kernels* à `https://github.com/gnuradio/volk`

b. G. Goavec-Merou, J.-M Friedt, *"On ne compile jamais sur la cible embarquée" : Buildroot propose GNU Radio sur Raspberry Pi (et autres)*, Hackable **37** (Avril-Mai-Juin 2021)

c. `https://limemicro.com/sdr/limesdr-xtrx/`

Mises en garde

- **Stabilité** du traitement numérique du signal, mais inexactitude de la représentation en virgule flottante : $\sin(2\pi \cdot f \cdot (t/N)) \neq \sin(2\pi \cdot (f/N) \cdot t)$
- Nombreuses dépendances de GNU Radio : pérennité ? Graphe des dépendances sous Debian GNU/Linux :



Matériel pour la radio logicielle

- RTL-SDR : faible coût, bande passante modeste
- Ettus Research (racheté NI) : schémas disponibles [5]

Nom	Porteuse [6]	BW [7] (MHz)	Rés. [8]	Émetteur [9]	Coût (Dec. 2024)
RTL-SDR	50-1600	2,4	8 bits	N	9 euros
Pluto (AD)	70-6000 (325–3800)	20 MHz	12 bits	O	270 euros
HackRF One	1-6000	20 MHz	8 bits	O	305 euros
LimeSDR	0,1-3800	61,4 MHz	12 bits	O MIMO	379$
BladeRF	300-3800	40 MHz	12 bits	O	540$
Ettus B210	70-6000	61,4 MHz	12 bits	O MIMO	2220 euros
Ettus X310	70-6000	160 MHz	12 bits	O MIMO	9680 euros+frontend

5. `https://files.ettus.com/schematics/`
6. Définit la bande de fréquence accessible, e.g. 88–108 MHz FM commerciale, 137 MHz sat. POES, 460 MHz POCSAG, 1574,2 MHz GPS, 1620 MHz Iridium ...
7. Détermine la nature des protocoles décodables
8. Dynamique $20 \log_{10}(256) = 48$ dB ; $20 \log_{10}(4096) = 72$ dB
9. L'émission est règlementée sur les bandes de fréquence, puissance et rapport cyclique

Conclusion

GNU Radio pour le prototypage rapide et la mise en œuvre de radio logicielle

- opensource pour apprendre du code et en corriger les erreurs
- (cross-)compilable sur toute plateforme
- bibliothèques C++ liées à Python par PyBIND
- forum de discussion `https://chat.gnuradio.org/#/room/#gnuradio:gnuradio.org` et liste d'échanges `https://lists.gnu.org/mailman/listinfo/discuss-gnuradio`
- videos (présentations et conférences European GNU Radio days) : `https://www.youtube.com/@europeangnuradiodays1445`

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM