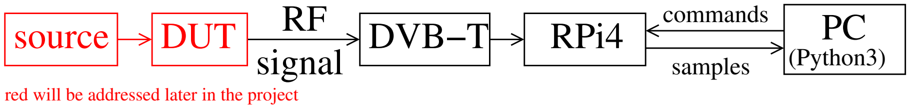
Outline

General context: we wish to design an instrument in which the data are collected by the Raspberry Pi 4, under control of the PC, to be transferred to the PC for processing and display.



GNU Radio on Raspberry Pi 4

1. making sure GNU Radio is properly installed [1]: accessing GNU Radio blocks and playing a sound
2. first demonstration with RTL-SDR dongle: FM receiver
3. from RPi4 to PC used as sound card: Zero-MQ publish/subscribe
4. from PC to RPi4: TCP/IP server running as a Python thread

Objective: a FM radio receiver running on the RPi4, streaming sound from the RPi4 to the PC, whose carrier frequency is controlled from the PC

[1] `raspberrypi4_64_gnuradio_defconfig` Buildroot configuration file with GNU Radio support (and dependencies) at `https://github.com/oscimp/oscimp_br2_external/tree/master/configs`

Challenge: consistency between PC and embedded system

We will generate GNU Radio flowcharts on the PC, convert to Python script and execute on the RPi4

Currently **three** generations of GNU Radio incompatible with each other: 3.7 (obsolete since 2019 [2], uses XML), 3.8 (YAML, SWIG, Python3, Qt5 [3]), 3.9/3.10 (YAML, no SWIG $\rightarrow$ Pybind11 [4])

GNURadio 3.8 (Debian stable, Ubuntu 20)

- default configuration in Buildroot 2022.11 archive
- can benefit from `https://github.com/oscimp/oscimp_br2_external/blob/master/configs/raspberrypi4_64_gnuradio_defconfig`

GNURadio 3.9/10 (Debian testing/unstable, Ubuntu 22)

- requires git version of Buildroot at `https://git.buildroot.net/buildroot` (will become Buildroot archive in 02/2023)

Alternatively, compile a custom known version of GNU Radio using PyBOMBS (`https://github.com/gnuradio/pybombs`) **after removing all binary distribution packages**

[2] Debian oldstable/Ubuntu 18

[3] `https://wiki.gnuradio.org/index.php/GNU_Radio_3.8_OOT_Module_Porting_Guide`

[4] `https://wiki.gnuradio.org/index.php/GNU_Radio_3.9_OOT_Module_Porting_Guide`

Buildroot with GNU Radio support

GNU Radio requires multiple additional options not selected with the default Buildroot:

- **glibc** C library (instead of uClibc)
- **eudev** device handling
- Python3 support
- some additional GNU Radio options (Python support, 0-MQ ...)

to avoid iterative selection of the Buildroot packages, a new **defconfig** file is available from

`https://github.com/oscimp/oscimp_br2_external/tree/master/configs`

Download [5] **raspberrypi4_64_gnuradio_defconfig**, put the file in the local Buildroot **configs**, and restart the whole compilation

```
make clean
make raspberrypi4_64_gnuradio_defconfig
make
```

(should be faster since the downloaded archives are still in **dl/**)
Total disk space: about 12 GB

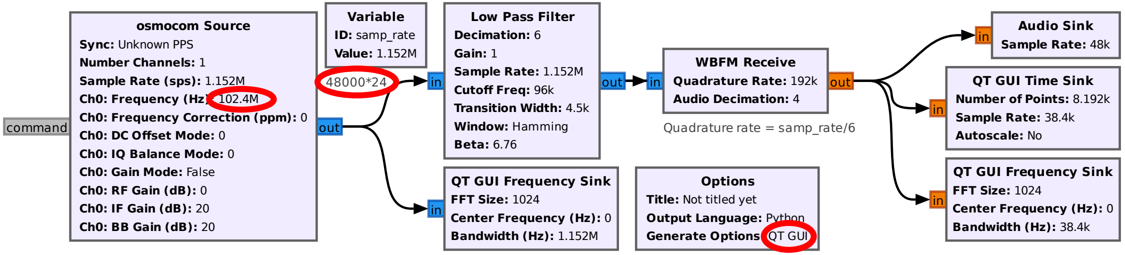
[5] make sure to download the `raw` (top-right menu) file, and not its HTML formatted code

GNU Radio 3.8 on the PC (only if binary distribution version >3.8)

- `pybombs` allows for multiple version of GNU Radio to be installed on a same computer (including 3.8, 3.9 and 3.10)
- We will launch `gnuradio-companion` on **the PC**:
  1. check the packaged version of GNU Radio: `apt-cache policy gnuradio`
  2. if **not** the wanted version, use PyBOMBS (Python Build Overlay Managed Bundle System) as described at `https://github.com/gnuradio/pybombs`
     - 2.1 `sudo apt-get install python3-pip`
     - 2.2 `sudo pip3 install pybombs`
     - 2.3 `pybombs auto-config`
     - 2.4 `pybombs recipes add-defaults`
     - 2.5 `pybombs prefix init ~/prefix-3.8 -R gnuradio38` assuming the installation directory is in `$HOME/prefix-3.8`
     - 2.6 `source ~/prefix-3.8/setup_env.sh` (will have to be repeated in each terminal launching GNU Radio Companion)
     - 2.7 `gnuradio-companion`
  3. if GNU Radio Companion has started: install osmosdr, either from the binary package apt if GNU Radio 3.8 is packaged, or with PyBOMBS (`pybombs install gr-osmosdr`) if this installation system was used
  4. close GNU Radio Companion and launch again (or refresh package list) to access Osmocom Source: the links between blocks must be curves ($\geq$3.8) and not lines at right angles ($\leq$ 3.7).

GNU Radio on PC: broadcast FM station reception

1. RTL-SDR sample rate $\in [1.0 : 2.4]$ MS/s (`samp_rate`)
2. Osmocom Source: carrier frequency ($\in [25 : 1600]$ MHz, $\in [88 : 108]$ MHz for broadcast FM)
3. bandpass filter to select a single FM station (250 kHz banwidth) + decimation
4. WBFM receiver[6]
5. decimate the WBFM output to reach one of the available audio sampling rates (e.g. 48 kHz) assuming `samp_rate` was wisely selected)
6. assess (`QT Freq Sink` spectrum analyzer) each processing step using the graphical interface features
7. dynamically change the carrier frequency using a GUI slider setting the variable



osmocom Source
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 1.152M
Ch0: Frequency (Hz): 102.4M
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Variable
ID: samp_rate
Value: 1.152M

48000*24

Low Pass Filter
Decimation: 6
Gain: 1
Sample Rate: 1.152M
Cutoff Freq: 96k
Transition Width: 4.5k
Window: Hamming
Beta: 6.76

WBFM Receive
Quadrature Rate: 192k
Audio Decimation: 4

Quadrature rate = samp_rate/6

Audio Sink
Sample Rate: 48k

QT GUI Time Sink
Number of Points: 8.192k
Sample Rate: 38.4k
Autoscale: No

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 1.152M

Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 38.4k

[6] D. Bederov, *Arithmetic based implementation of a quadrature FM Demodulator*, FOSDEM (2015) at `https://archive.fosdem.org/2015/schedule/event/sdr_arithmetic/`

GNU Radio on Raspberry Pi4

1. Check that GNU Radio is properly installed: on the RPi4,

   ```
   # python3
   import gnuradio
   ```

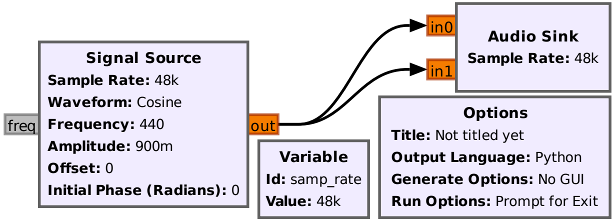
   must return with a prompt and no warning/error
2. basics of GNU Radio flowcharts: one source, digital processing blocks, and sinks with consistent datarate along the processing path (**samp_rate**/decimation factors)
3. no graphical output in the Raspberry Pi4: launch `gnuradio-companion` on the PC and select **Options** → **Generate Options** → **No GUI**
4. the **Id** defines the name of the output Python script
5. **Run** → **Generate** to convert the flowgraph in a Python script (see console for output file & path)
6. copy (**scp** [7] the Python script from the PC to the Raspberry Pi4
7. on the Raspberry Pi4, execute with **python3 my_script.py**

[7] on the PC: `scp my_script.py root@pi4_IP_address:/root`

GNU Radio on Raspberry Pi4

1. the trivial flowchart generated on the PC with GNU Radio Companion (3.8 – check the curved connections)
   - 2 audio outputs (GNU Radio inputs) for stereo
   - 48 kS/s sampling rate
   - sine wave signal source
   - run on the RPi4 with **python3 pgm.py**

   ⇒ must output a tone on the audio jack



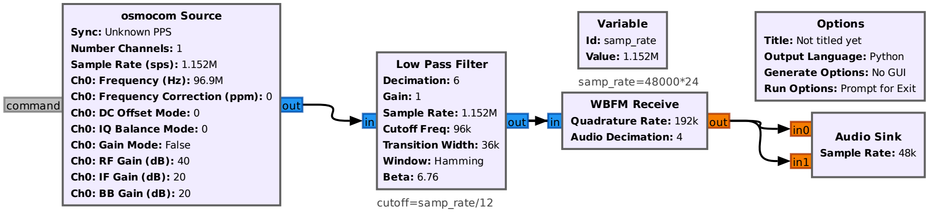
Signal Source
Sample Rate: 48k
Waveform: Cosine
Frequency: 440
Amplitude: 900m
Offset: 0
Initial Phase (Radians): 0

Audio Sink
Sample Rate: 48k

Variable
Id: samp_rate
Value: 48k

Options
Title: Not titled yet
Output Language: Python
Generate Options: No GUI
Run Options: Prompt for Exit

2. FM radio receiver to check proper operation of DVB-T dongle



osmocom Source
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 1.152M
Ch0: Frequency (Hz): 96.9M
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 40
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Low Pass Filter
Decimation: 6
Gain: 1
Sample Rate: 1.152M
Cutoff Freq: 96k
Transition Width: 36k
Window: Hamming
Beta: 6.76

cutoff=samp_rate/12

Variable
Id: samp_rate
Value: 1.152M

samp_rate=48000*24

WBFM Receive
Quadrature Rate: 192k
Audio Decimation: 4

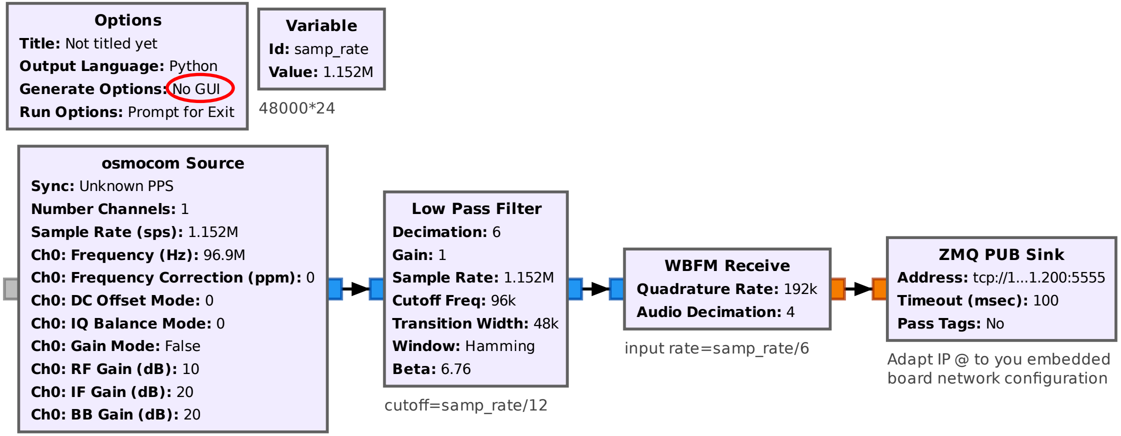
Options
Title: Not titled yet
Output Language: Python
Generate Options: No GUI
Run Options: Prompt for Exit

Audio Sink
Sample Rate: 48k

Streaming from RPi4 to PC

- An instrument collects the data and sends them to a PC for processing
- the PC might not process all data but only segments
- UDP-like Zero-MQ stream: **Publish-Subscribe** mechanism (supported by Python, GNU/Octave, C, C++ ...)
- demonstration: stream the FM demodulated sound to the PC used as sound card.

On the Raspberry Pi4, fetch samples, demodulate and send



Options
Title: Not titled yet
Output Language: Python
Generate Options: No GUI
Run Options: Prompt for Exit

Variable
Id: samp_rate
Value: 1.152M

48000*24

osmocom Source
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 1.152M
Ch0: Frequency (Hz): 96.9M
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 10
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20

Low Pass Filter
Decimation: 6
Gain: 1
Sample Rate: 1.152M
Cutoff Freq: 96k
Transition Width: 48k
Window: Hamming
Beta: 6.76

cutoff=samp_rate/12

WBFM Receive
Quadrature Rate: 192k
Audio Decimation: 4

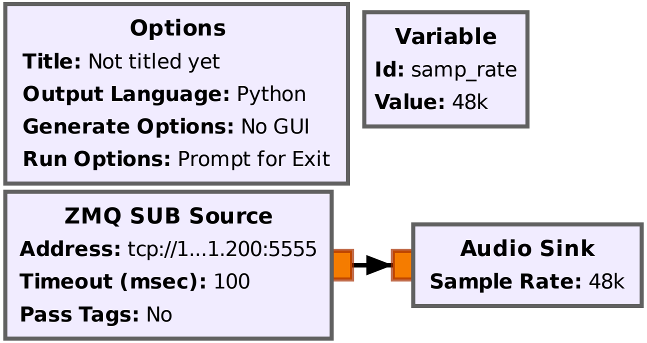
input rate=samp_rate/6

ZMQ PUB Sink
Address: tcp://1...1.200:5555
Timeout (msec): 100
Pass Tags: No

Adapt IP @ to you embedded board network configuration

Streaming from RPi4 to PC

On the PC:



Options
Title: Not titled yet
Output Language: Python
Generate Options: No GUI
Run Options: Prompt for Exit

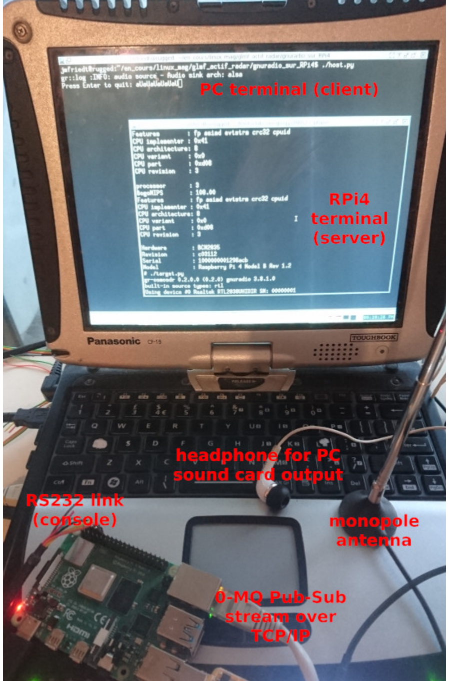
Variable
Id: samp_rate
Value: 48k

ZMQ SUB Source
Address: tcp://1...1.200:5555
Timeout (msec): 100
Pass Tags: No

Audio Sink
Sample Rate: 48k

IP is the embedded board network address

set the Subscribe Source to the **tcp://192.168.x.y:5555** IP address of the Raspberry Pi4, same port as before.



```
jmfriedt@rugged:~/en_cours/linux_mag/glmf_actif_radar/gnuradio_sur_RPi4$ ./host.py
gr::log :INFO: audio source - Audio sink arch: alsa
Press Enter to quit: aUaUaUaUaUaU
```

```
Features        : fp asimd evtstrm crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd08
CPU revision    : 3

processor       : 3
BogoMIPS        : 108.00
Features        : fp asimd evtstrm crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd08
CPU revision    : 3

Hardware        : BCM2835
Revision        : c03112
Serial          : 100000001298acb
Model           : Raspberry Pi 4 Model B Rev 1.2
# ./target.py
gr-osmosdr 0.2.0.0 (0.2.0) gnuradio 3.8.1.0
built-in source types: rtl
Using device #0 Realtek RTL2838UHIDIR SN: 00000001
```

Commands from PC to RPi4

Multithreaded Python script approach

- GNU Radio Companion is a Python script generator
- GNU Radio Companion 3.8 allows for inserting additional Python commands in its initialization code: **Python Snippets**
- GNU Radio Companion 3.8 allows for adding Python functions: **Python Module**
- Launch a separate thread running a TCP (connected mode) server
- Receive commands from the PC running a TCP client (**telnet**)
- Tune the GNU Radio flowgraph variables by calling the callback function associated with the modified variable
- Alternate ZeroMQ solution: REQ/REP (Request/Reply)

What is a thread ?

- function run in parallel to the main program but sharing the same memory space

```
import threading
import time

def jmf1(argument):
        while True:
                print(argument)
                time.sleep(1)

threading.Thread(target=jmf1,args=(1,)).start()
threading.Thread(target=jmf1,args=(2,)).start()
threading.Thread(target=jmf1,args=(3,)).start()
```

- make sure to kill/quit the thread before leaving the Python script (`self.my_status=...`)

What is a server ?

Definition: a *server* waits for a connection, a *client* connects to the server when it needs information [8]

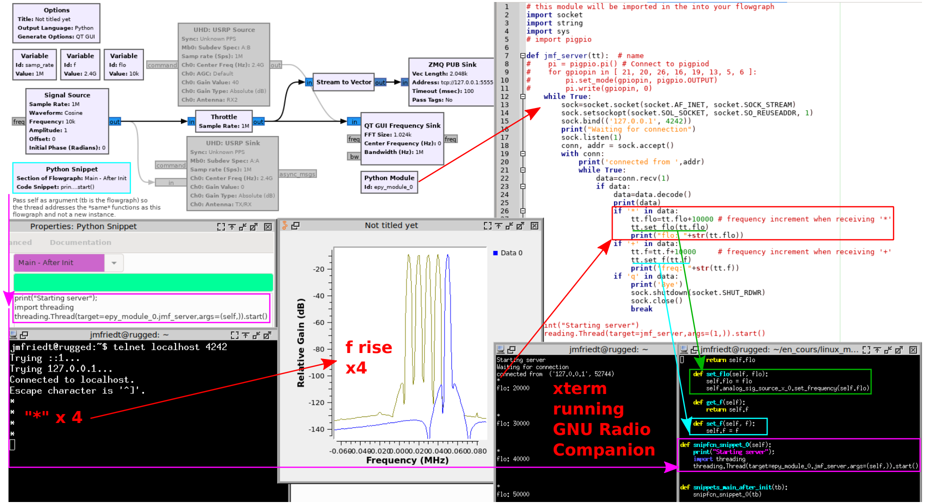
```
import socket
import string
while True:
    sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('127.0.0.1', 4242))
    print("Waiting for connection")
    sock.listen(1)
    conn, addr = sock.accept()
    with conn:
        print('connected from ',addr)
        while True:
            data=conn.recv(1)
            if data:
                data=data.decode()
                print(data)
                if 'q' in data:
                    sock.shutdown(socket.SHUT_RDWR)
                    sock.close()
                    break
```

- Run `python3 my_server` in one terminal
- Run `telnet localhost 4242` in another terminal
- Enjoy ... quit by sending 'q'

[8] availability of the selected port can be checked using `nmap localhost` which lists ports used by running services

Putting it all together ...

Python Snippet executes the thread including the Python Module running the TCP server controlling the GNU Radio execution by tuning parameters with the associated callback function [9]



Options — Title: Not titled yet; Output Language: Python; Generate Options: QT GUI

Variable — Id: samp_rate; Value: 1M

Variable — Id: f; Value: 2.4G

Variable — Id: flo; Value: 10k

UHD: USRP Source — Sync: Unknown PPS; Mb0: Subdev Spec: A:B; Samp rate (Sps): 1M; Ch0: Center Freq (Hz): 2.4G; Ch0: AGC: Default; Ch0: Gain Value: 40; Ch0: Gain Type: Absolute (dB); Ch0: Antenna: RX2

Signal Source — Sample Rate: 1M; Waveform: Cosine; Frequency: 10k; Amplitude: 1; Offset: 0; Initial Phase (Radians): 0

Throttle — Sample Rate: 1M

Stream to Vector

ZMQ PUB Sink — Vec Length: 2.048k; Address: tcp://127.0.0.1:5555; Timeout (msec): 100; Pass Tags: No

QT GUI Frequency Sink — FFT Size: 1.024k; Center Frequency (Hz): 0; Bandwidth (Hz): 1M

UHD: USRP Sink — Sync: Unknown PPS; Mb0: Subdev Spec: A:A; Samp rate (Sps): 1M; Ch0: Center Freq (Hz): 2.4G; Ch0: Gain Value: 0; Ch0: Gain Type: Absolute (dB); Ch0: Antenna: TX/RX

Python Module — Id: epy_module_0

Python Snippet — Section of Flowgraph: Main - After Init; Code Snippet: prin....start()

Pass self as argument (tb is the flowgraph) so the thread addresses the *same* functions as this flowgraph and not a new instance.

Properties: Python Snippet — Main - After Init

```
print("Starting server");
import threading
threading.Thread(target=epy_module_0.jmf_server,args=(self,)).start()
```

```python
# this module will be imported in the into your flowgraph
import socket
import string
import sys
# import pigpio

def jmf_server(tt):  # name
#    pi = pigpio.pi() # Connect to pigpiod
#    for gpiopin in [ 21, 20, 26, 16, 19, 13, 5, 6 ]:
#        pi.set_mode(gpiopin, pigpio.OUTPUT)
#        pi.write(gpiopin, 0)
    while True:
        sock=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        sock.bind(('127.0.0.1', 4242))
        print("Waiting for connection")
        sock.listen(1)
        conn, addr = sock.accept()
        with conn:
            print('connected from ',addr)
            while True:
                data=conn.recv(1)
                if data:
                    data=data.decode()
                    print(data)
                    if '*' in data:
                        tt.flo=tt.flo+10000 # frequency increment when receiving '*'
                        tt.set_flo(tt.flo)
                        print("flo: "+str(tt.flo))
                    if '+' in data:
                        tt.f=tt.f+10000     # frequency increment when receiving '+'
                        tt.set_f(tt.f)
                        print("freq: "+str(tt.f))
                    if 'q' in data:
                        print('Bye')
                        sock.shutdown(socket.SHUT_RDWR)
                        sock.close()
                        break

int("Starting server")
reading.Thread(target=jmf_server,args=(1,)).start()
```

```
jmfriedt@rugged:~$ telnet localhost 4242
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
*
*
*
*
```

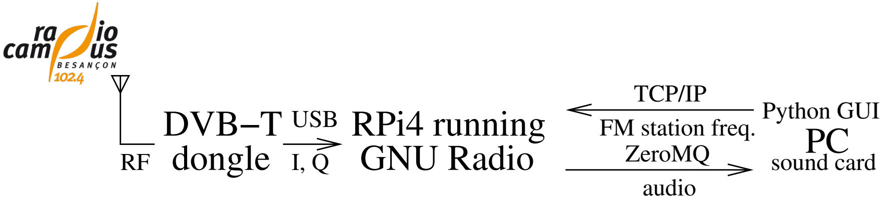
```
Starting server
Waiting for connection
connected from  ('127.0.0.1', 52744)
*
flo: 20000


*
flo: 30000


*
flo: 40000


*
flo: 50000
```

```
        return self.flo

    def set_flo(self, flo):
        self.flo = flo
        self.analog_sig_source_x_0.set_frequency(self.flo)

    def get_f(self):
        return self.f

    def set_f(self, f):
        self.f = f

def snipfcn_snippet_0(self):
    print("Starting server");
    import threading
    threading.Thread(target=epy_module_0.jmf_server,args=(self,)).start()


def snippets_main_after_init(tb):
    snipfcn_snippet_0(tb)
```

[9] J.-M Friedt, W. Feng, *Analyse et réalisation d'un RADAR à synthèse d'ouverture (SAR) par radio logicielle (2/3)*, GNU/Linux Magazine France 242 (Nov. 2020)

Commands from PC to RPi4

- Demonstrate how you modify the previous flowchart, streaming the output of the FM demodulator to the PC, to tune the broadcast station frequency on the Raspberry Pi4 from the PC.
- Provide a graphical user interface allowing to enter the FM radio frequency and transferring the information to the RPi4



Why GNU Radio $>$3.8?

Link between Python and C++:

- $\leq$3.8: SWIG (Simplified Wrapper and Interface Generator) – runtime errors (dynamic library linking issues)
- $\geq$3.9[10]: C++(11/14) bindings through Pybind11 – compilation errors [11]

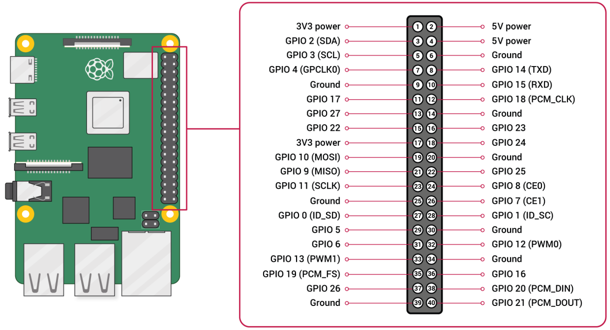
[10] D. Estévez, *GNU Radio 3.9 in Buildroot* at https://destevez.net/2021/10/gnu-radio-3-9-in-buildroot/

[11] https://wiki.gnuradio.org/index.php/GNU_Radio_3.9_OOT_Module_Porting_Guide

Next week

Design a dedicated signal source based on the AD9954 DDS:

1. read datasheet, identify necessary pins and passive components, analyze evaluation board
2. match signals with those available on the Raspberry Pi4 40-pin connector[12] including power supply and communication signals
3. schematic: logic relations between signals
4. board routing
5. mechanical design: connector location
6. Bill of Materials (BoM), supplier and manufacturing cost



[12] https://www.raspberrypi.com/documentation/computers/images/GPIO-Pinout-Diagram-2.png