Current issue with gr-osmosdr

cmake rule for gr-osmosdr master branch now requires GNU Radio 3.9.

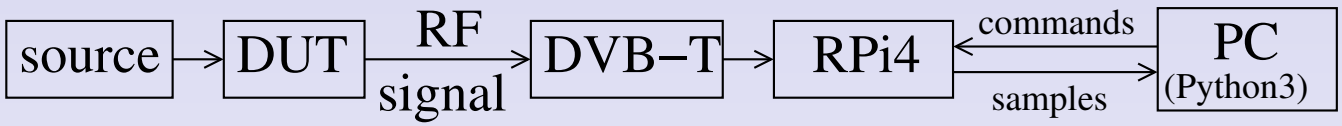
For GNU Radio 3.8:

- edit /.pybombs/recipes/gr-recipes/gr-osmosdr.lwr and replace gitbranch: master with gitbranch: gr3.8
- same modification in /.pybombs/recipes/gr-recipes/gr-iqbal.lwr
- clean src directory and pybombs install gr-osmosdr

Check https://github.com/osmocom/gr-osmosdr for available branches

Outline

General context: embedded network analyzer architectured around the Raspberry Pi 4 and using an RTL-SDR DVB-T dongle as radiofrequency receiver.



Emitting a radiofrequency signal from the Raspberry Pi 4 clock

1. Investigating radiofrequency emission sources
2. Using the RPi4 internal PLL feeding a GPIO as radiofrequency source
3. Making sure the radiofrequency is controlled and understood by receiving with the DVB-T dongle

Objective: emitting an FM radio signal from the Raspberry Pi4 and listening to the resulting sound [1]

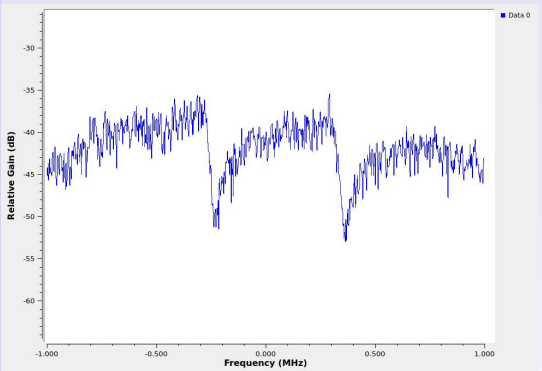
[1] sample video of expected result: `http://jmfriedt.free.fr/201229_rpitx.mp4`

Radiofrequency sources

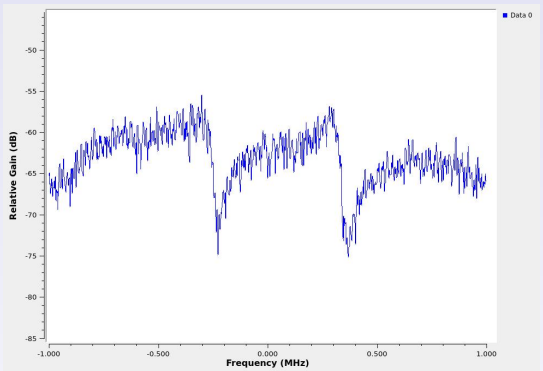
Characterize the transfer function of a passive Device Under Test
$\Rightarrow$ radiofrequency driving signal

- broadband = noise: Zener diode, but requires high (24 V) voltage for broadband signal + radiofrequency amplifiers
- pulse: must be short and sharp edges. Test with ADCMP fast comparators (e.g. ADCMP573 [2] for single supply operation): functional but requires an external trigger, e.g. RPi PWM

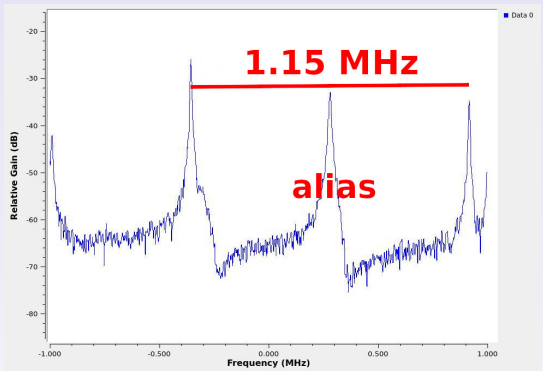
These solutions require additional, external hardware and are prone to artefacts ...



Broadband noise source



40 ns pulse every 160 ns



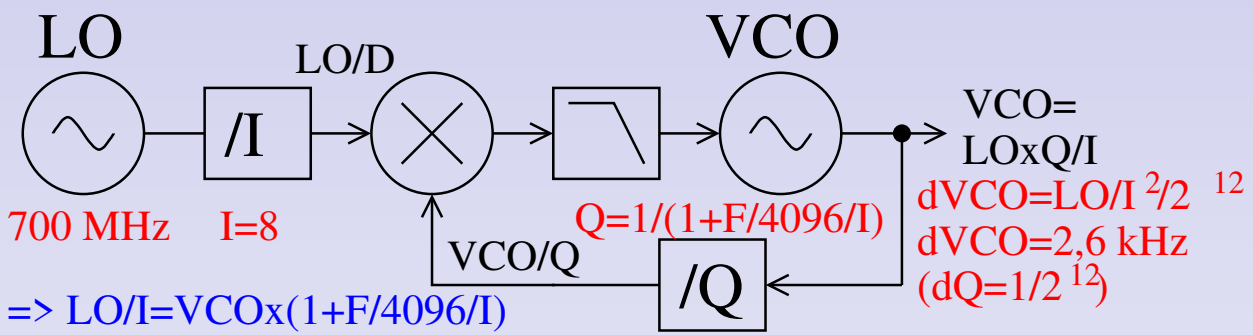
40 ns pulse every 800 ns

... but the RPi GPIO can be driven from a radiofrequency clock source ! See the PiFM project [3].

[2] https://www.analog.com/en/products/adcmp573.html

[3] http://www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter

Fractional PLL



- Raspberry Pi single board computers provide a reference clock LO (700 MHz for RPi4, 500 MHz for others)
- LO feeds a fractional Phase Locked Loop (PLL [4]) with a pre-scaler of $I \in \mathbb{N}$
- the PLL Voltage Controlled Oscillator (VCO) is divided by $Q \in \mathbb{Q}$
- the phase comparator compares LO/I with VCO/Q:

  $VCO = LO \times Q/I = LO/I \times \left(1 + \frac{F}{I \times 4096}\right)^{-1}$ (see [5])$= LO/\left(I + F/4096\right)$
- output frequency < 125 MHz (GPIO limitation [6]) $\Rightarrow$ use overtone (5th overtone of FM band to reach 434 MHz ISM band)
- output frequency resolution: considering that $\boldsymbol{VCO = LO \times Q/I}$ and that the resolution $\boldsymbol{dF}$ on $\boldsymbol{F \in [0:4095]}$ is **1**, **what is the frequency resolution at 434 MHz? Is it enough for our application? How does it compare with a DDS?**

[4] https://elinux.org/The_Undocumented_Pi#Clocks

[5] https://datasheets.raspberrypi.org/bcm2711/bcm2711-peripherals.pdf p.81 "5.4 General Purpose GPIO Clock"

[6] https://datasheets.raspberrypi.org/bcm2711/bcm2711-peripherals.pdf p.82 "5.4.1 Operating Frequency"

Fractional PLL

Many implementations derived from the original PiFM demonstration [7]:

- `https://github.com/ChristopheJacquet/PiFmRds` is easiest [8] to understand
- GPIO clock sourced from a fractional PLL is described pp.104–105 of `https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0.pdf`
- a much more general (and complex [9]) implementation is available at `github.com/F5OEO/rpitx` relying on `github.com/F5OEO/librpitx`
- interfacing the latter with GNU Radio {I,Q} stream is explained at `https://github.com/ha7ilm/rpitx-app-note`
- `http://abyz.me.uk/rpi/pigpio/pigs.html` explains that "Access to clock 1 is protected by a password as its use will likely crash the Pi. The password is given by or'ing 0x5A000000 with the GPIO number."

Our application only requires a single continuous-wave (CW) tone for a Frequency Swept CW analyzer (FSCW)

---

[7] O. Mattos & O. Weigl, `https://github.com/rm-hull/pifm` described at `http://www.icrobotics.co.uk/wiki/index.php/Turning_the_Raspberry_Pi_Into_an_FM_Transmitter`

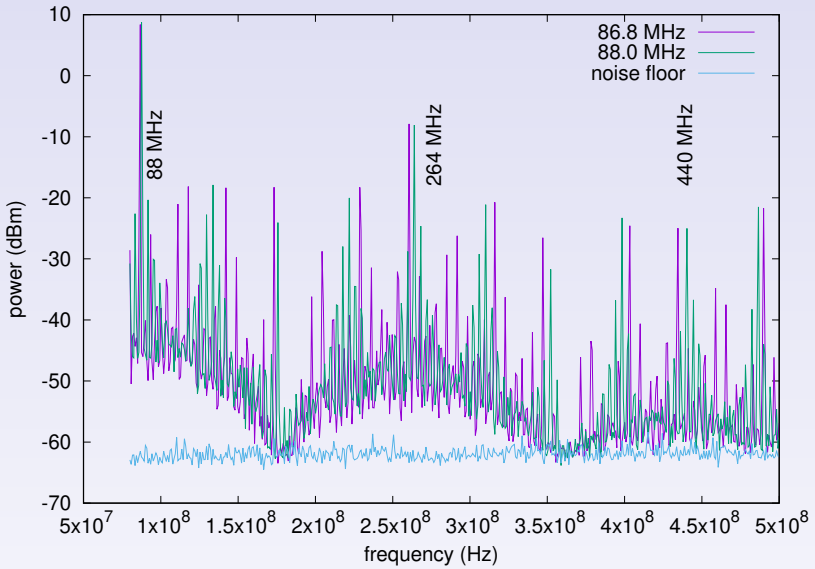
[8] `github.com/ChristopheJacquet/PiFmRds/blob/master/src/pi_fm_rds.c#L534`

[9] E. Courjaud *Rpitx: Raspberry Pi SDR transmitter for the masses*, SDRA (2017) at `https://www.youtube.com/watch?v=Jku4i8t_nPc`

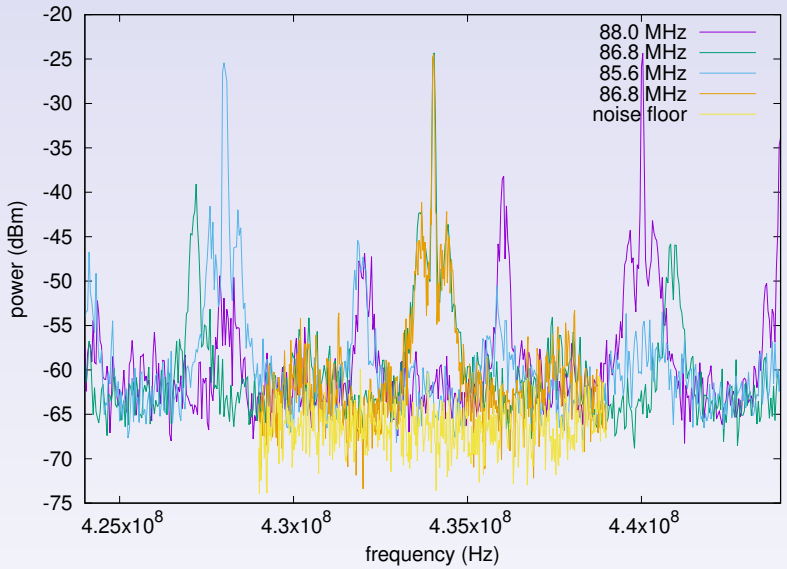
Overtone

The RPi GPIO has been observed to generate a strong signal up to 250 MHz.

We aim for the 434 MHz band $\Rightarrow$ use overtones



Broadband spectrum

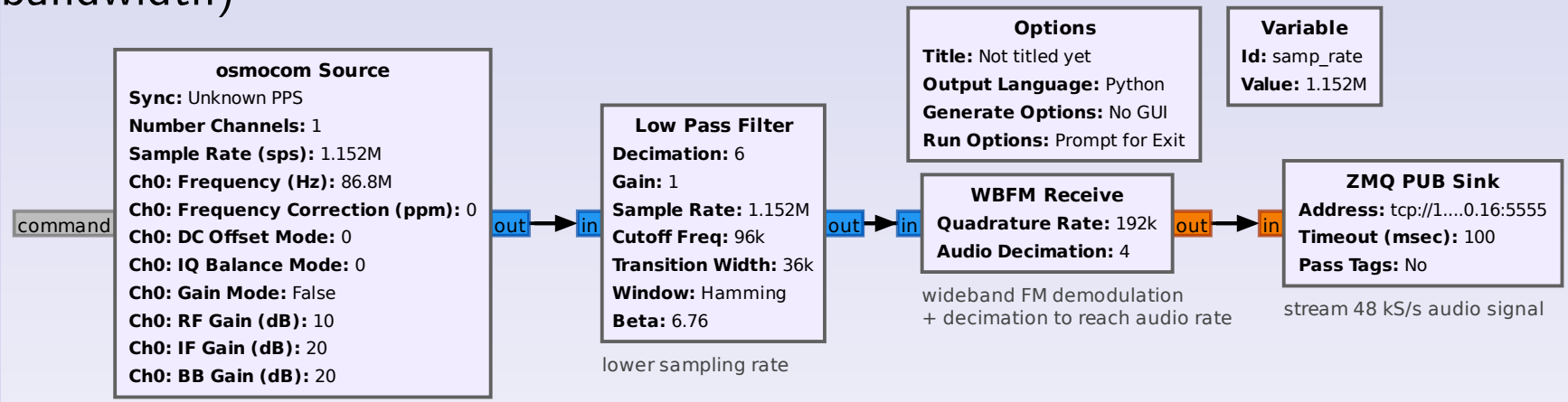


Zoom on the (FM modulated) overtone signal

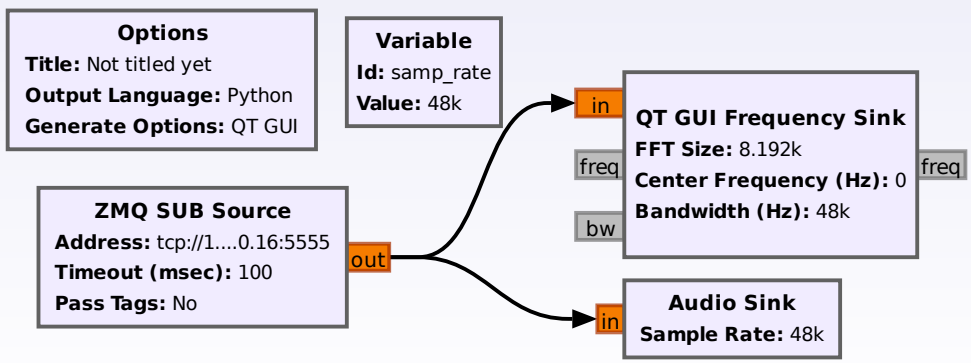
Square wave output $\Rightarrow$ overtone N scales as $1/N$. Emit at $434/5 = 86.8$ MHz

FM emission/reception from the RPi4

On the **embedded board**: CLI flowchart for acquisition, demodulation and streaming (lowering the sampling rate and hence the communication bandwidth)



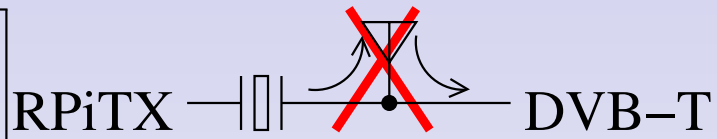
On the **host PC**: GUI for displaying the spectrum and playing audio on the sound card



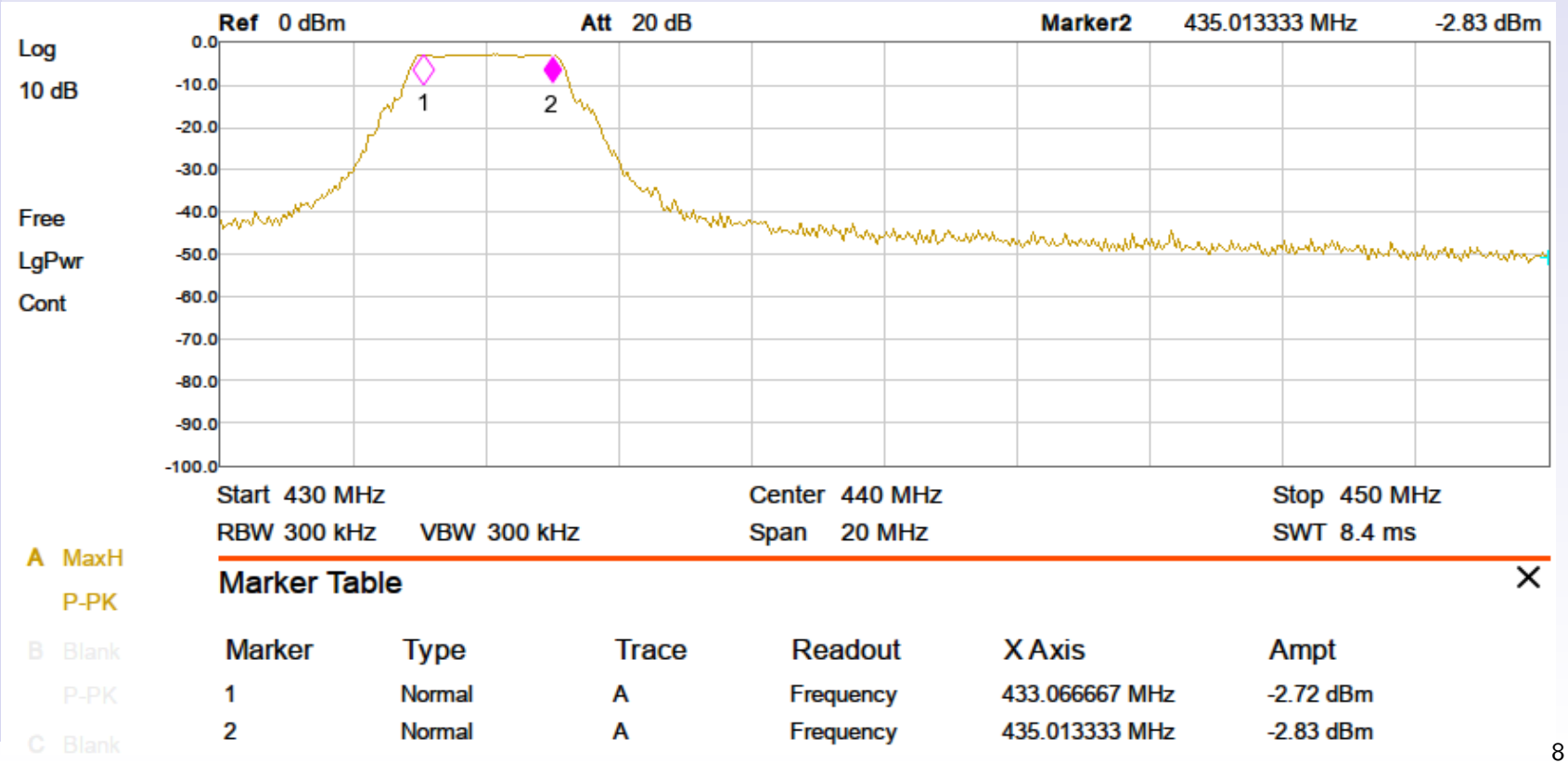
SAW resonator characterization

The SAW device is connected in parallel to the antenna ⇒

make sure to disconnect the antenna (MCX connector) when measuring the SAW device to avoid radiating the probe signal and collecting environmental noise !



Example of radiated spectrum when seeping around the 434 MHz ISM band:



Conclusion

`PiFM` is provided as Buildroot external package (`BR2_EXTERNAL`) at `https://github.com/oscimp/PlutoSDR/` in the `for_next` branch

1. Compile PiFM and its dependenciens for the RPi4
2. Generate signals and check that their spectra are consistent with expectations (tone, chirp, FM)
3. Listen to the generated signals
4. Control the emitted signal, in addition to the received signal, from the Python server

Do not add a wire to GPIO4: the pin will radiate a low enough power to be detected by the DVB-T receiver without bothering neighbours

Next step: characterize the SAW resonator transfer function

```
# ./NA.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
Detached kernel driver
Found Rafael Micro R820T tuner
[R82XX] PLL not locked!
[R82XX] PLL not locked!
Allocating 15 zero-copy buffers
Press Enter to quit:
```

RPi4 RX

```
# ./pi_fm_rds -freq 86.92 -intval 10
Using mbox device /dev/vcio.
Allocating physical memory: size = 3403776     mem_ref = 4     bus_addr = fe8bb000     virt_addr = 0x7fb9449000
divider is 1644.7368 (1644 + 3018*2^-12) [nominal 1096.4912].
Starting to transmit on 86.920 MHz.
```

RPi4 TX