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
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

1sample video of expected result: http://jmfriedt.free.fr/201229_rpitx.mp4
Radiofrequency sources

Characterize the transfer function of a passive Device Under Test
⇒ 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 ...

---

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

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\(^3\).
Fractional PLL

- Raspberry Pi single board computers provide a reference clock $LO$ (700 MHz for RPi4, 500 MHz for others)
- this clock feeds a fractional Phase Locked Loop (PLL \(^4\)) with a pre-scaler of $D$
- the PLL Voltage Controlled Oscillator (VCO) is divided by $N$
- the phase comparator compares $LO/D$ with $VCO/N$: $VCO = LO \times N/D$
- output frequency < 200 MHz (GPIO limitation) ⇒ use overtone (5th overtone of FM band to reach 434 MHz ISM band)
- output frequency resolution: considering that $VCO = LO \times N/D$ and that the resolution $dD$ on $D$ is $2^{-12}$, 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
Fractional PLL

Many implementations derived from the original PiFM demonstration ⁵:

- https://github.com/ChristopheJacquet/PiFmRds is easiest ⁶ 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 ⁷) 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)

---


⁶ github.com/ChristopheJacquet/PiFmRds/blob/master/src/pi_fm_rds.c#L534

⁷ E. Courjaud Rpitx: Raspberry Pi SDR transmitter for the masses, SDRA (2017) at https://www.youtube.com/watch?v=Jku4i8t_nPc
The RPi GPIO has been observed to generate a strong signal up to 250 MHz.

We aim for the 434 MHz band ⇒ use overtones

Broadband spectrum

Zoom on the (FM modulated) overtone signal

Square wave output ⇒ 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)

- **osmocom Source**
  - **Sync**: Unknown PPS
  - **Number Channels**: 1
  - **Sample Rate (sps)**: 1.152M
  - **Ch0**: **Frequency (Hz)**: 86.8M
  - **Ch0**: **Frequency Correction (ppm)**: 0
  - **Ch0**: **DC Offset 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**: 36k
  - **Window**: Hamming
  - **Beta**: 6.76

- **WBFM Receive**
  - **Quadrature Rate**: 192k
  - **Audio Decimation**: 4

- **ZMQ PUB Sink**
  - **Address**: tcp://1.1.1.1:5555
  - **Timeout (msec)**: 100
  - **Pass Tags**: No

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

- **QT GUI Frequency Sink**
  - **FFT Size**: 8.192k
  - **Center Frequency (Hz)**: 0
  - **Bandwidth (Hz)**: 48k

- **Audio Sink**
  - **Sample Rate**: 48k

---

**Variable**
- **Id**: samp_rate
- **Value**: 1.152M
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 dependencies 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