ANALOG DEVICES

AHEAD OF WHAT'S POSSIBLE™

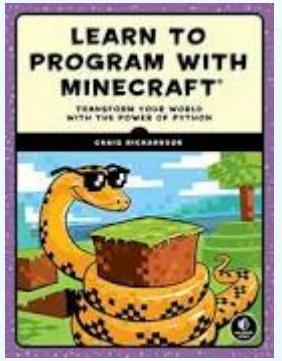
Python for the Rest of Us

Presented by Mark Thoren

Virtual GRCon20

Recommended Preparation

- If you've never touched Python before, head straight to ***learnpython.org*** and go through the "Learn the Basics" lessons. Now. Seriously. It doesn't get any easier than this, programs actually run, but in the browser, no installation necessary.
- If you prefer paper, ***Python for Kids*** by Jason R. Briggs is surprisingly good for adults, too.
- As usual, Google, YouTube, Udemy, etc. are your friends. Find something that suits your style.



- Get either an ADALM2000 or ADALM Pluto, follow installation directions.
  - Install Putty (https://www.putty.org/) or use your favorite terminal, log in to 192.168.2.1 (user: root pw: analog)
  - Run iio_info and observe.
  - Run "**ps | grep iiod**" to see the thing that lets you connect remotely
  - On your local machine, open a command prompt, run "**iio_info**" and watch it fail
  - Run "**iio_info –n 192.168.2.1**" and compare!

- A fairly complete exercise on bringing up RPi + ADXL345:
  - https://wiki.analog.com/university/labs/software/iio_intro_toolbox
- Github repo for code / flowgraphs in this session:
  - https://github.com/mthoren-adi/gnuradio_projects/tree/grcon_2020

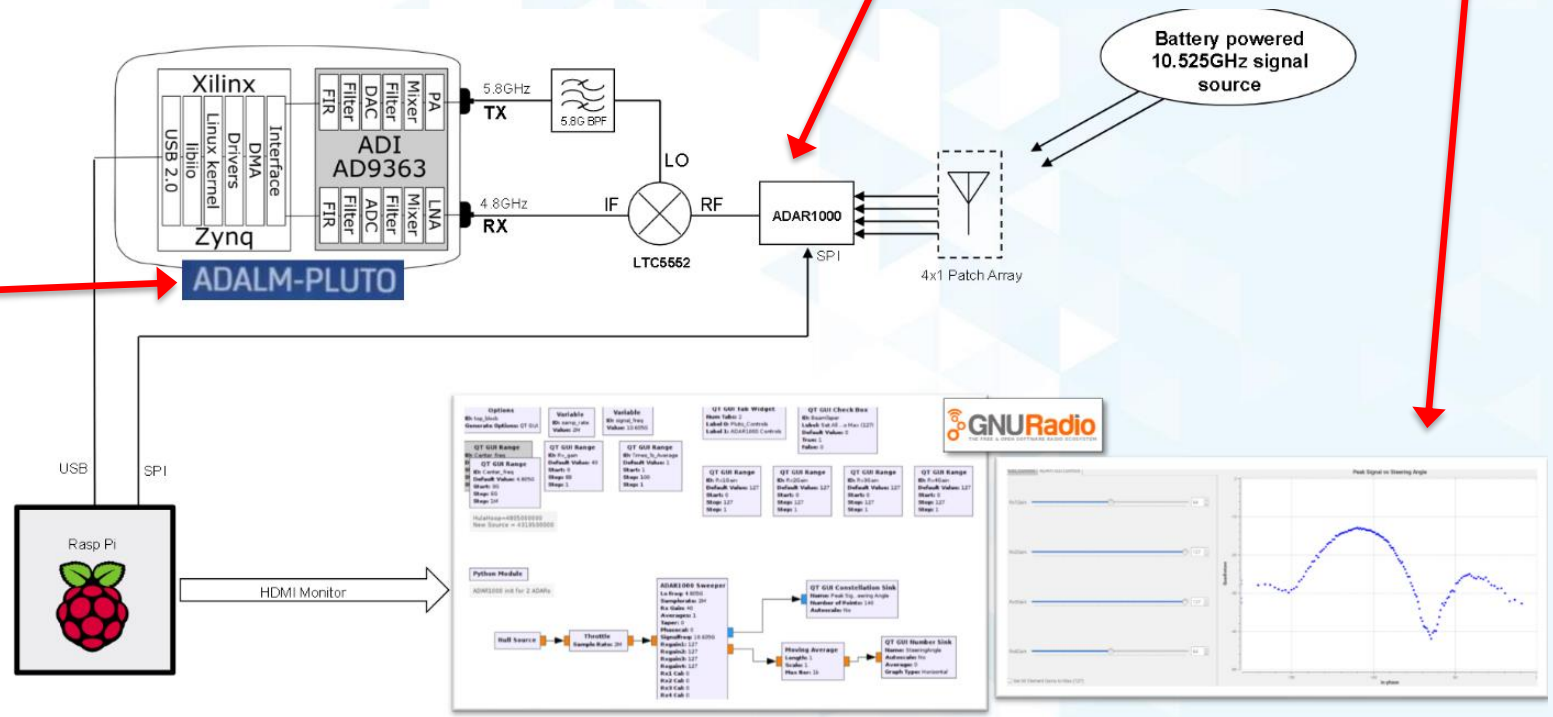
Motivation: Uses for “raw” Python along side GNURadio applications

- Automation, characterization, integrating GR with lab equipment
- Devices not supported in GNURadio
- Working around GR block limitations (partial / incomplete hardware interfaces)
- Convenience
- GUIs

Move complicated GUIs outside of GR

No GNURadio support ☹

Direct GNURadio support ☺



Agenda

1. Overview of hardware – ADXL345 Pmod board, connections to the Raspberry Pi expansion header, Power, Network, etc.
2. Overview of what's on the ADI Kuiper Linux SD card – drivers precompiled into kernel, libiio / gr-iio pre-installed. Basically, everything to get you to the point where you can see the accelerometer from within GNU Radio.
3. Example "Hello (physical) World!" application: GnuRadioADXL345-o-scope (IIO Attribute blocks feeding QT Time Sink)
4. Using embedded Python blocks
    a) ADXL345 controlling VCOs as proof-of-concept
    b) Using libiio to communicate with devices from within block
    c) Establish communication to external Python programs
5. Getting around GR Block limitations with Python (ADALM2000)
6. Python interfaces to other "radio-ish" parts (ADAR1000 microwave beamformer, AD9166 DAC, ADF4371, etc.)

First set of exercises at learnpython.org

Learn the Basics

- Hello, World!
- Variables and Types
- Lists
- Basic Operators
- String Formatting
- Basic String Operations
- Conditions
- Loops
- Functions
- Classes and Objects
- Dictionaries
- Modules and Packages

Hello, World!

Python is a very simple language, and has a very straightforward syntax. It encourages programmers to program without boilerplate (prepared) code. The simplest directive in Python is the "print" directive - it simply prints out a line (and also includes a newline, unlike in C).

To print a string in Python 3, just write:

script.py

```
1  print("Hello, 2020 Global FAE Conference!")
```

IPython Shell

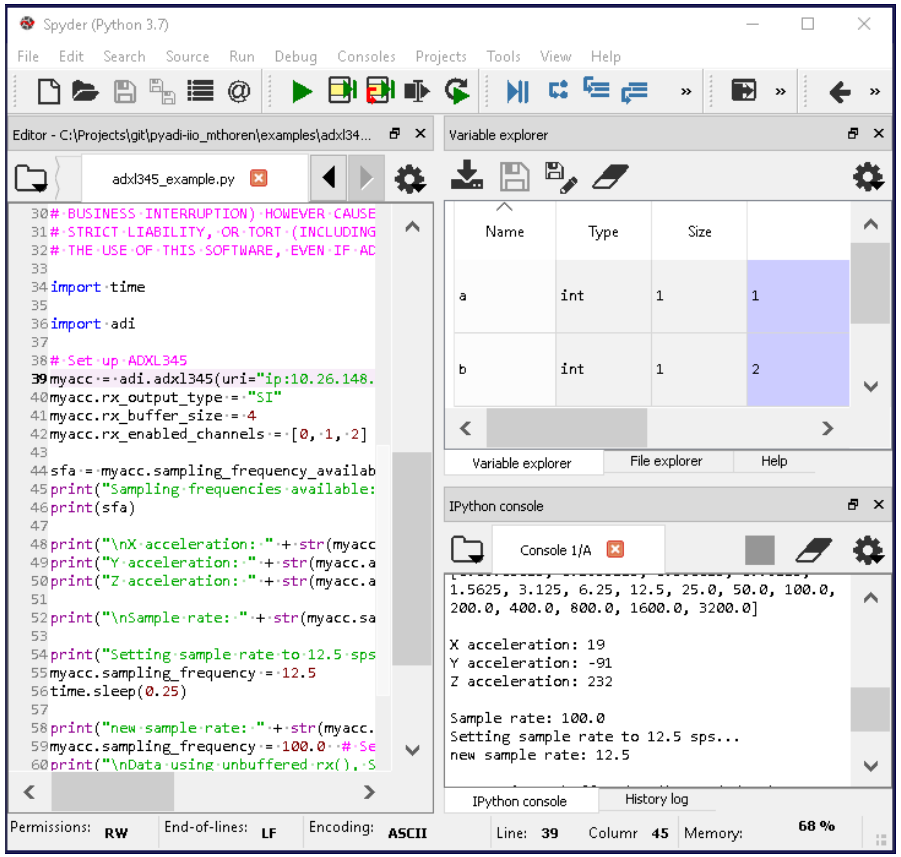
```
Hello, 2020 Global FAE Conference!

In [1]:
```

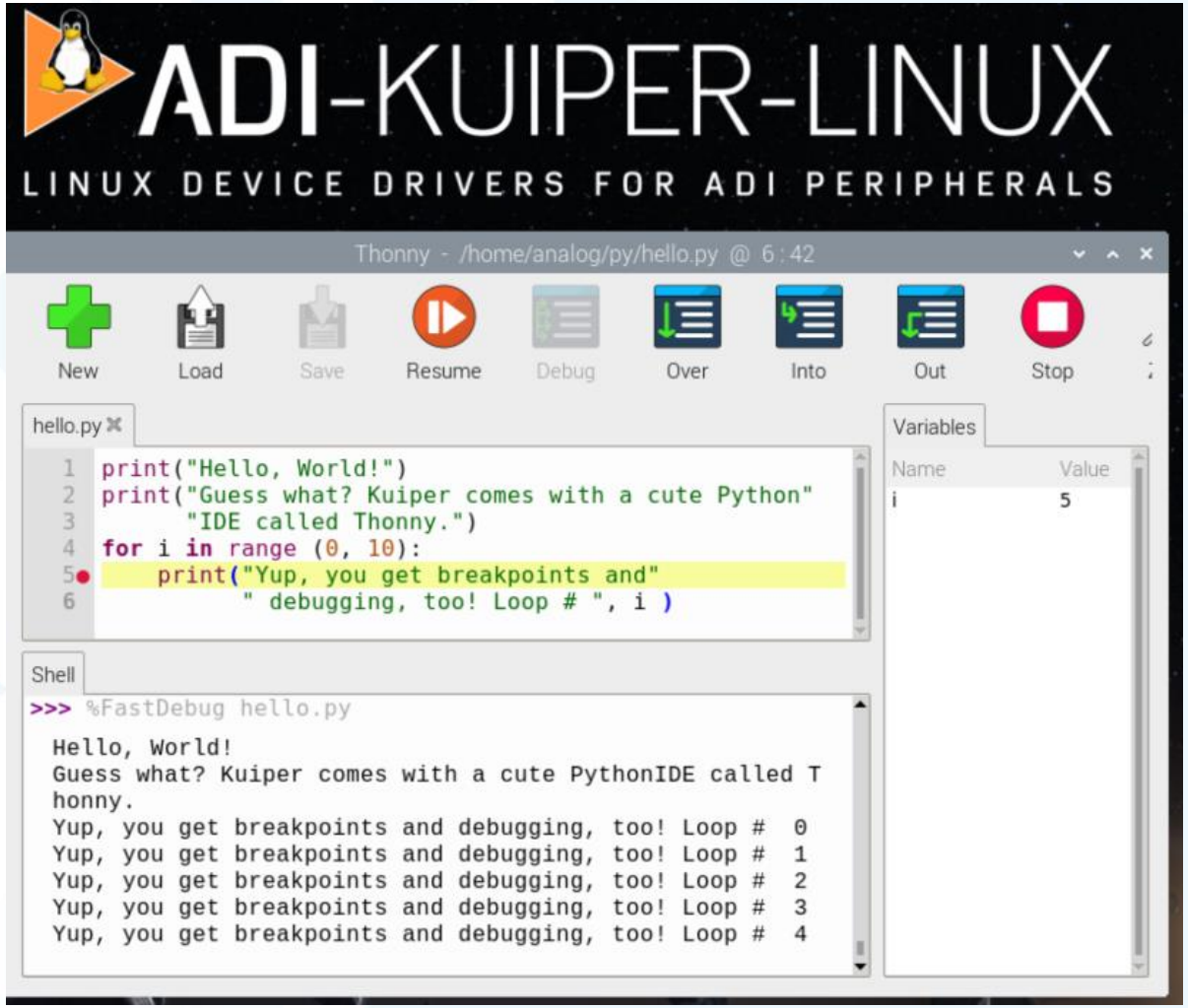
Run

Toolbox item: Python environment

- There are lots – largely a matter of preference.
  - Anaconda, PyCharm
  - “raw” installation (Mac, Linux)
  - Jupyter Notebook



Too many choices? No problem! Kuiper comes with one (Thonny)



ADXL345 + Rpi +GNURadio = World's Silliest Theremin*

Why ADXL345?

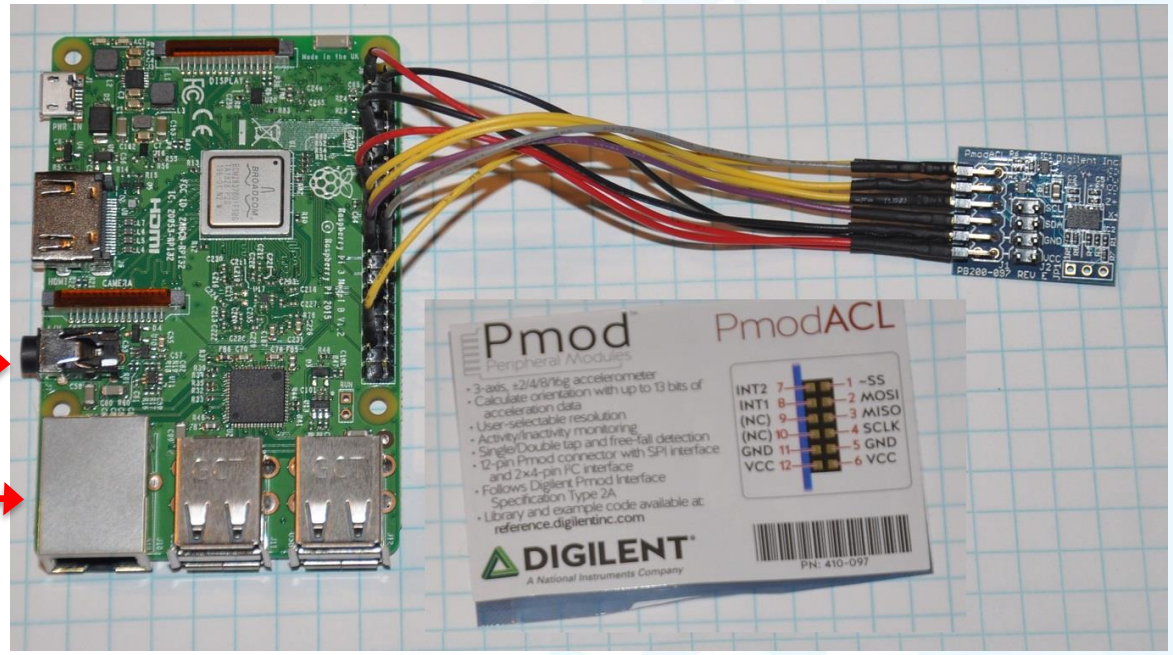
- Linux driver exists, and it's in the IIO framework
- Cheap Pmod
- Easy to connect w/ Schmartboard jumpers
- A human is a perfectly adequate signal source

Power (USB µ, C)

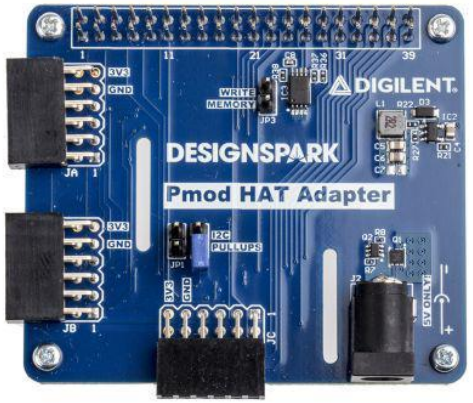
Display (optional if connecting via VNC)

Connect amplifier or headphones

Network, keyboard, mouse (Wireless works, too)



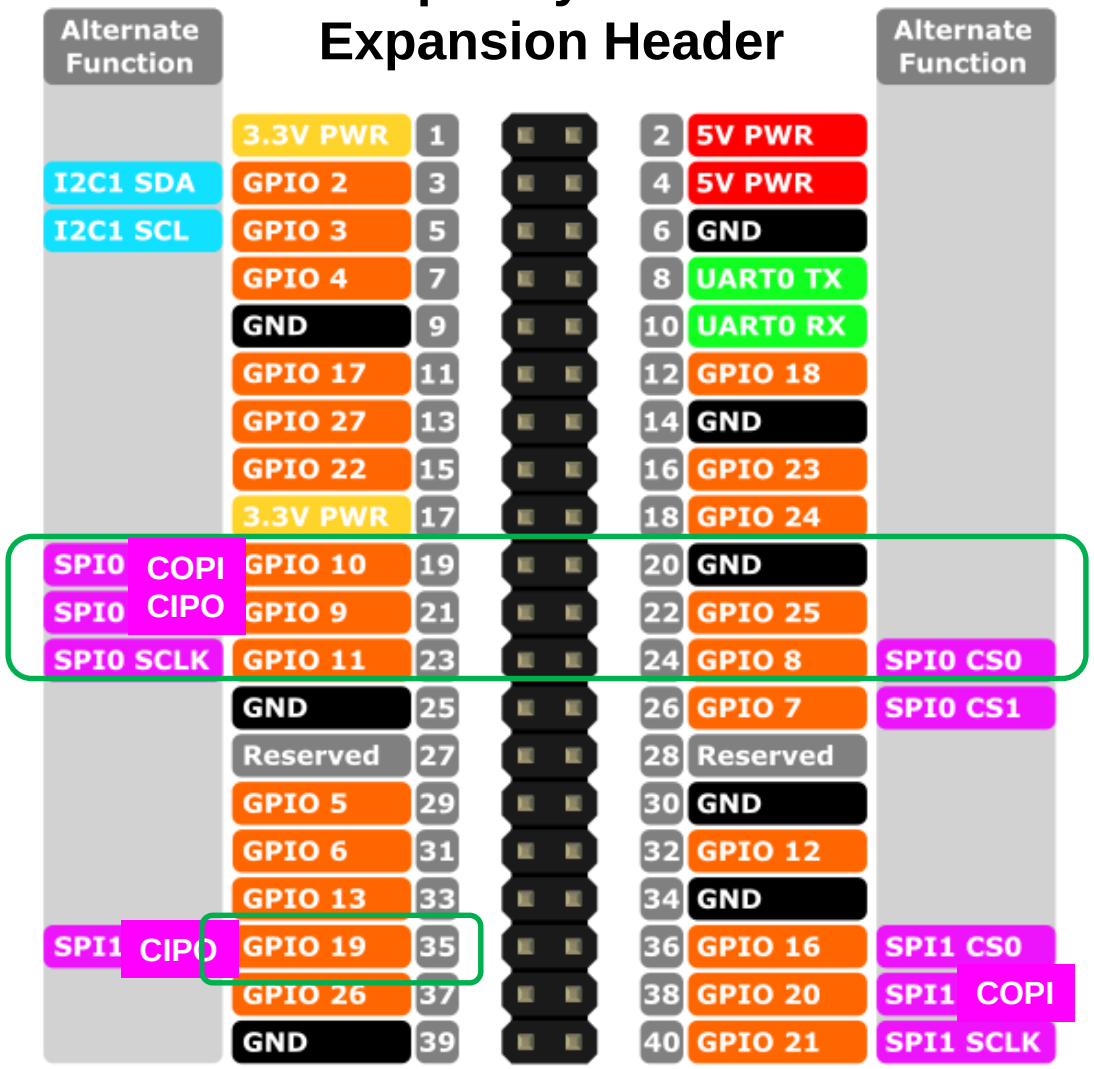
Alternate Hardware Connectivity option



*For the purists, Theremin-ish-like thingy.

Hardware Connections

Raspberry Pi Expansion Header



Pin Mapping:

GPIO19 ↔ INT2

CS0 ↔ ~SS

COPI ↔ COPI

CIPO ↔ CIPO

SCLK ↔ SCLK

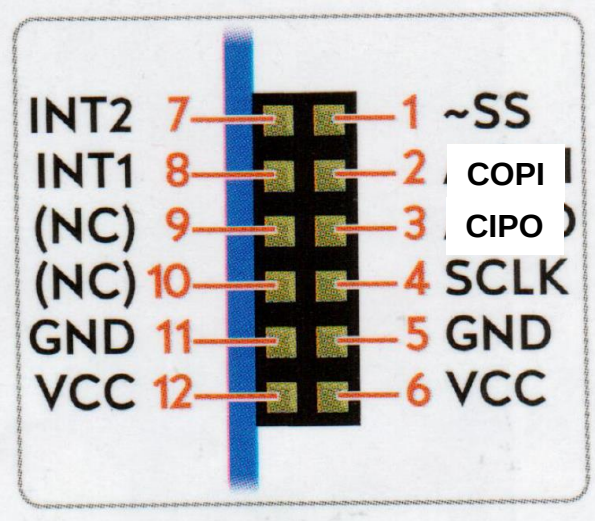
GND ↔ GND

GND ↔ GND

3V3 ↔ VCC

3V3 ↔ VCC

ADXL345 Pmod



(Connect w/ Schmartboard 5” female jumpers)

Toolbox Item: ADI-KUIPER-LINUX

LINUX DEVICE DRIVERS FOR ADI PERIPHERALS

- ADI's own variant of Raspbian!
- All appropriate Linux device drivers for ADI/LT parts enabled
  - Software libraries: the libiio, libm2k, pyadi-iio
  - GNURadio
  - IIO Oscilloscope

- Documentation / instructions: https://wiki.analog.com/resources/tools-software/linux-software/adi-kuiper_images
- Point config.txt to appropriate Device Tree Blob Overlay

For GRCon session (ADXL345): **`dtoverlay=rpi-adxl345`**

Pro Tip: run **`iio_info`** to check that connected hardware was found

What are “dtbo” and “dtb” files?

- **dtb** = device tree blob – information about hardware
  - Specific to a particular platform
- **dtbo** = device tree blob overlay, information about additional hardware (like the attached ADXL345)
  - Specific to a particular set of additional hardware (usually)
- **dts** = device tree source. The Device tree compiler (dtc) generates dtb,dtbo from these
- Where do they live?
  - Compiled examples on the SD card (boot partition)
  - Source in Linux repo – use these as a starting point if you need to make changes (ports, addresses, etc.)

Snippet from file:
`/linux/arch/arm/boot/dts/overlays/rpi-adxl345-overlay.dts`

```
fragment@0 {
        target = <&spi0>;
        __overlay__ {
                #address-cells = <1>;
                #size-cells = <0>;
                status = "okay";

                adxl345@0 {
                        compatible = "adi,adxl345";
                        reg = <0>;
                        spi-max-frequency = <1000000>;
                        spi-cpha;
                        spi-cpol;
                        interrupts = <19 IRQ_TYPE_LEVEL_HIGH>;
                        interrupt-parent = <&gpio>;
                };
        };
};
```

Toolbox Item: PyADI-IIO

PYTHON FOR ADI INDUSTRIAL I/O DEVICES

- From the Github Readme:

"pyadi-iio is a python abstraction module for ADI hardware with IIO drivers to make them easier to use."

- What pyadi-iio does for us – abstracts the gory details and "goofiness" of libiio, leaving us with a clean, Pythonic interface to our part.
- "glue layer" between iio (which has a bit of a learning curve) and doing something useful
- Pre-installed on ADI Kuiper Linux

Grab a chunk of data from an SDR chip in three** lines of code:

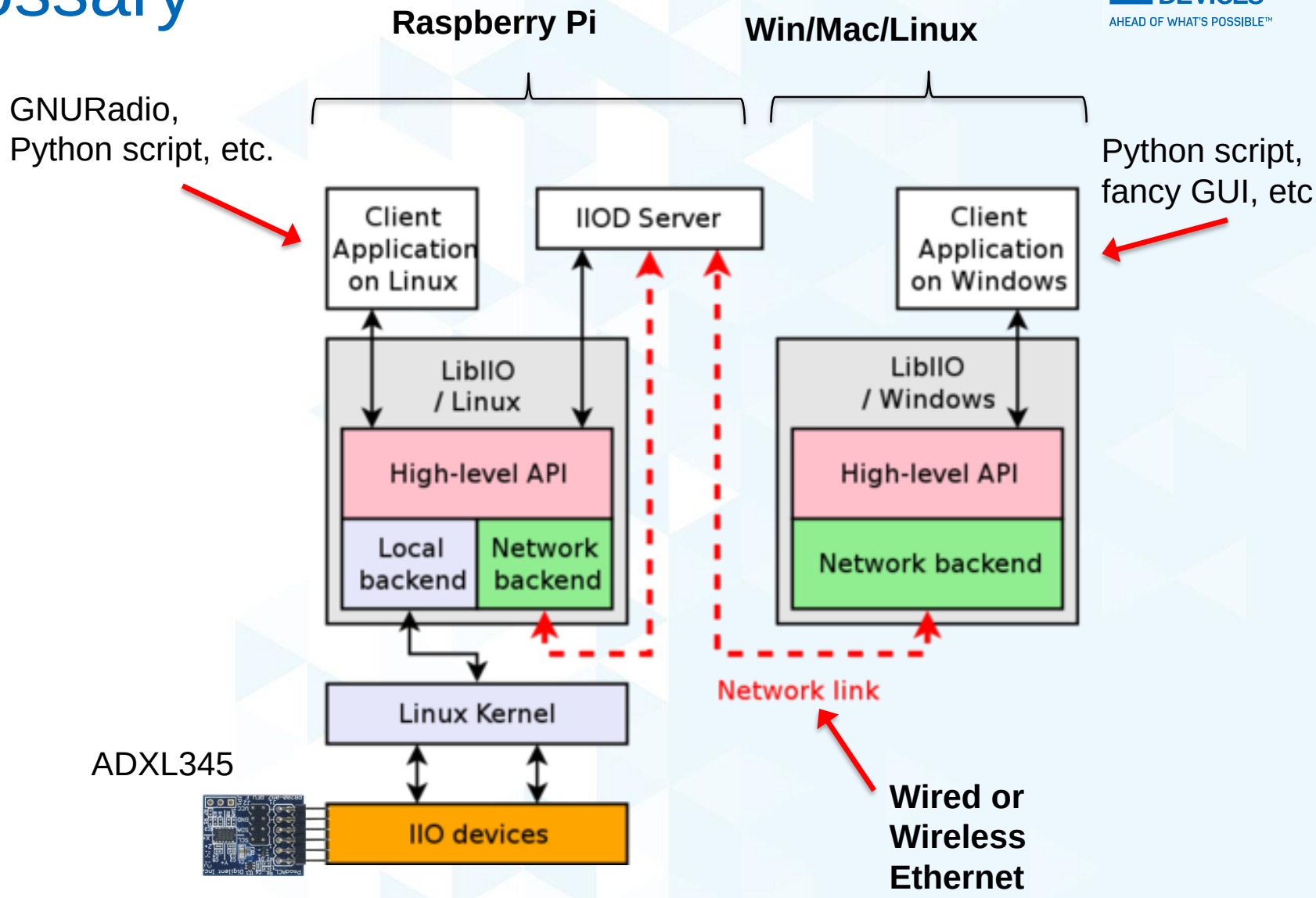
```
import adi

# Create device from specific uri address
sdr = adi.ad9361(uri="ip:192.168.2.1")
# Get data from transceiver
data = sdr.rx()
```

** Not counting comments ☺

Software Stack Glossary

- **IIO** – Industrial Input Output, a standard way of interfacing to data converters in Linux
- **libiio** – Library for talking to IIO devices from your program
- **IIOD** server – shares IIO devices over a network
- **Client application** – your (or customer's) program, can be written in pretty much any language.
- **Pyadi-iio** – If your client application is in Python, this makes your life easier



https://wiki.analog.com/resources/tools-software/linux-software/libiio

Pyadi-iio for the ADXL345

"`local:`" if script is running on your Rpi
"`ip:localhost`" standard user write access via iiod
"`ip:xxx.xxx.xxx.xxx`" to access remote target

Connect Device

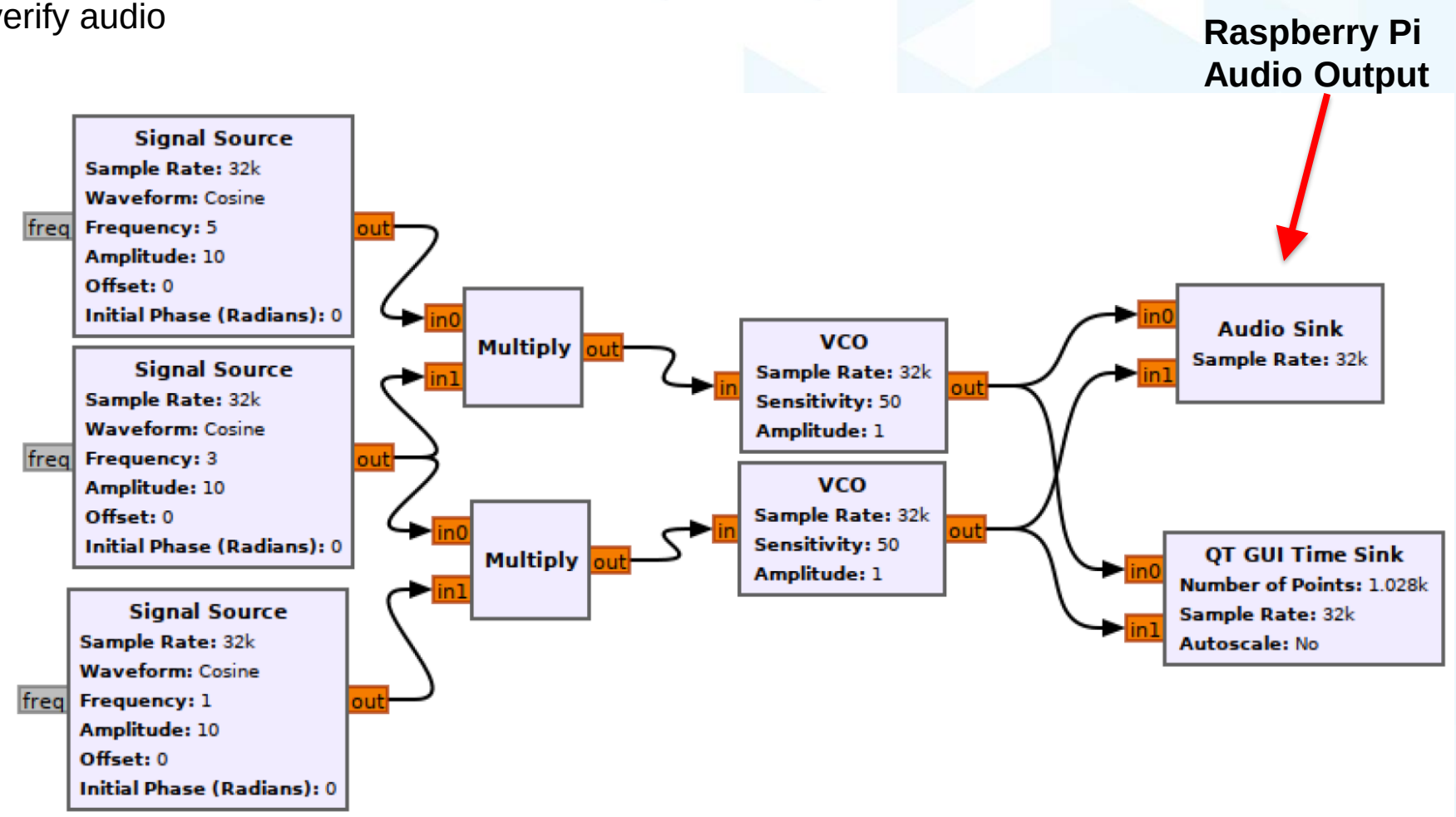
```
import adi
myacc = adi.adxl345(uri="ip:192.168.86.39")
myacc.sampling_frequency = 12.5
print("X acceleration: " + str(myacc.accel_x.raw))
```

Set sampling frequency

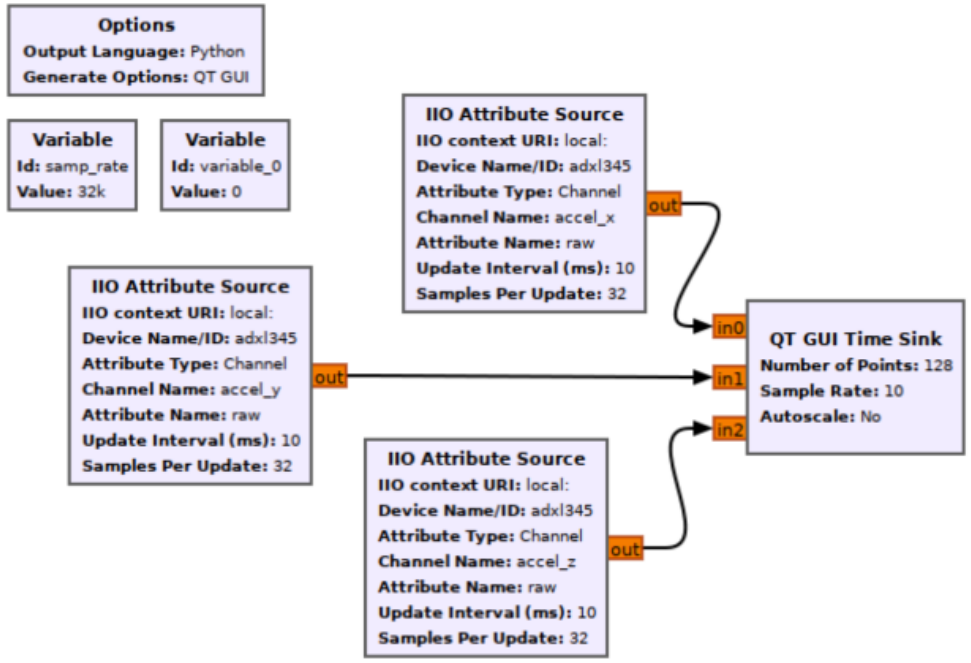
Read X acceleration

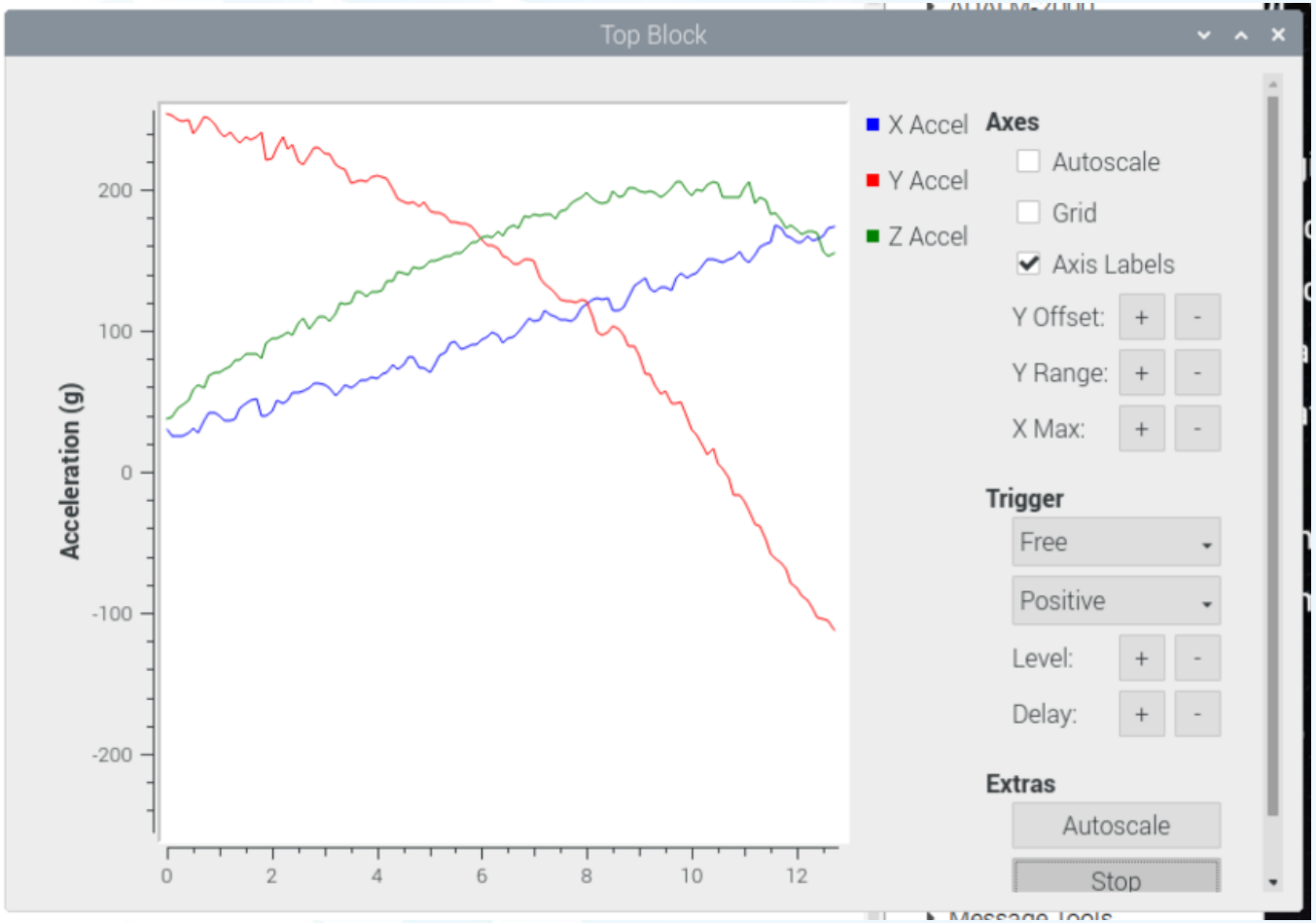
Hello, Audio: Trippy sound effect generator

- Main purpose is to verify audio functionality



Hello, Physical World: ADXL345-o-scope

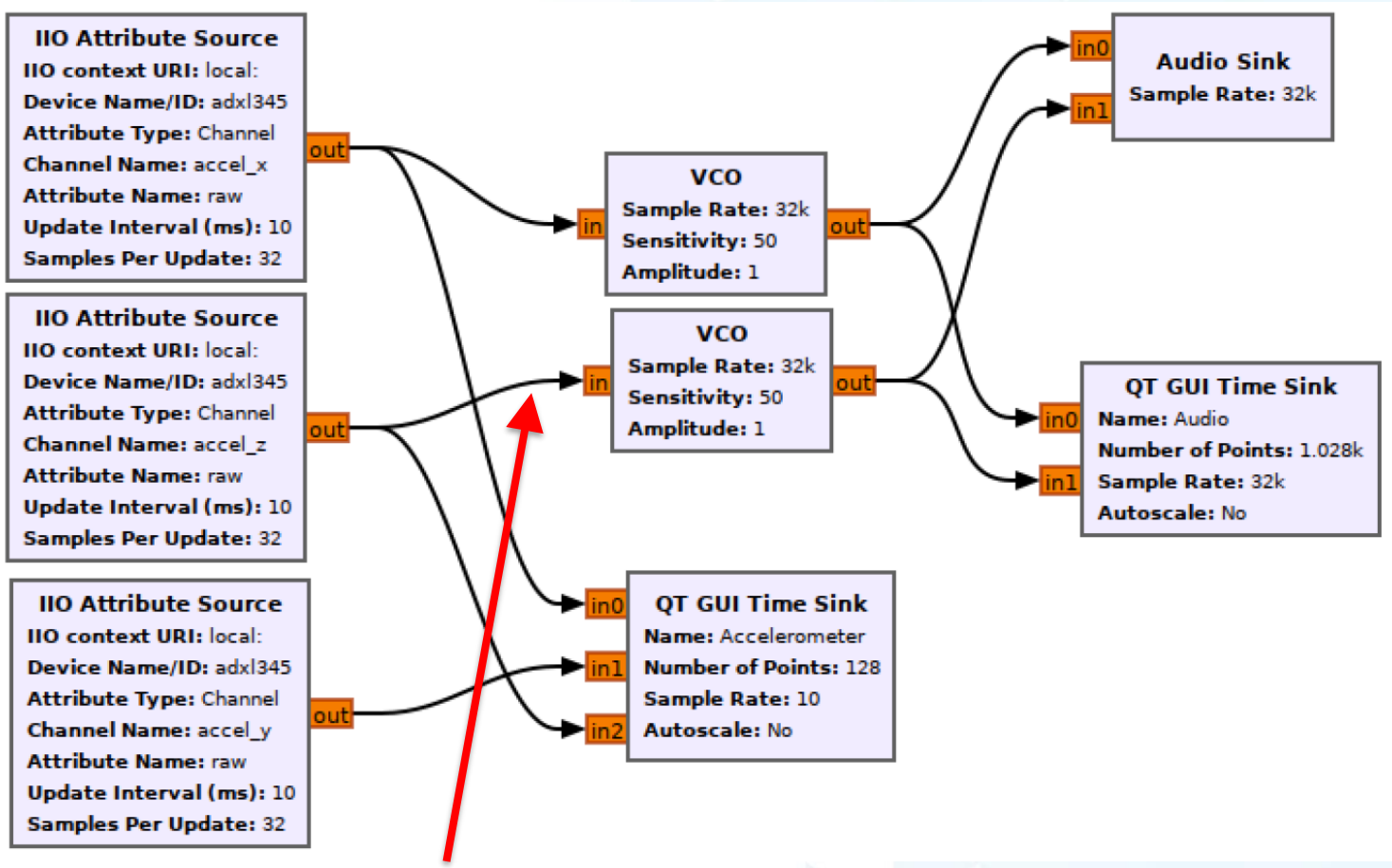




- Main purpose is to verify ADXL345-GR connectivity
- Uses GR IIO blocks directly

This doesn’t work

Well, mostly doesn’t. You get a little chirp every few seconds.



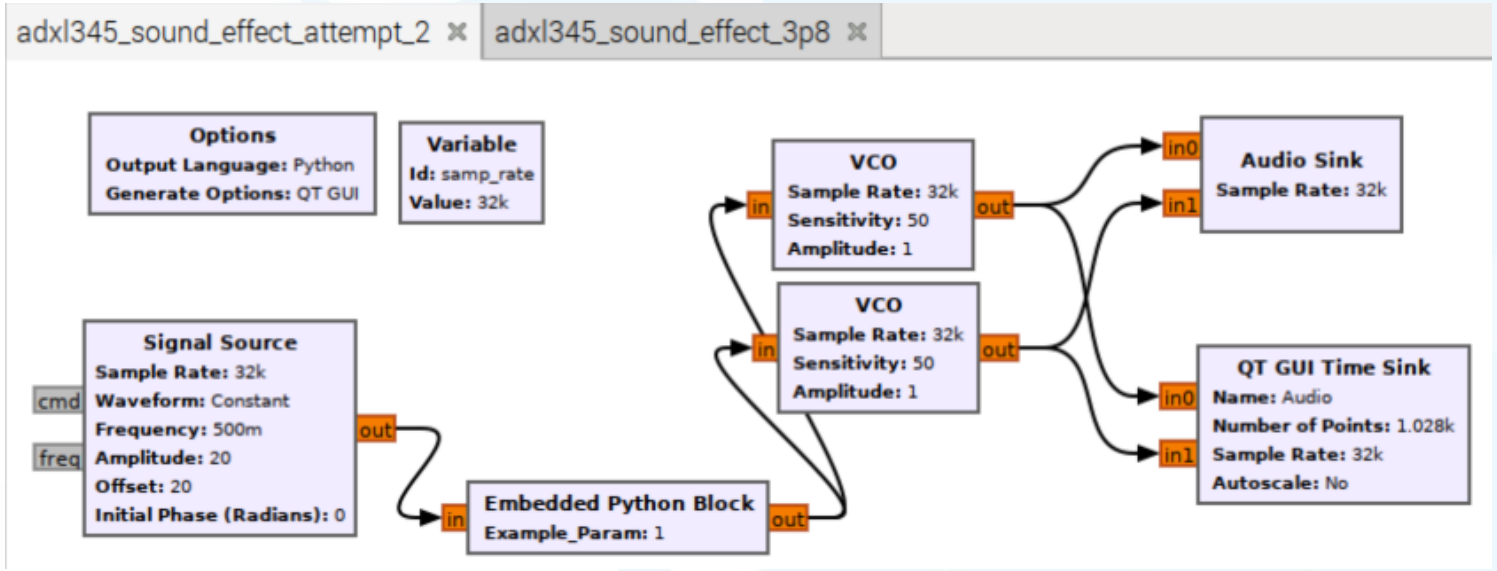
Fundamental mismatch… attribute vs. stream?

Sample and hold: Fail ☹

Rational Resampler: Fail ☹

Theremin*: Control VCOs from ADXL345

- Bridges gap between GR and outside world




```
9  import numpy as np
10 from gnuradio import gr
11 import adi
12
13 class blk(gr.sync_block):  # other base classes are basic_block, decim_bloc
14     """Embedded Python Block example - a simple multiply const"""
15
16     def __init__(self, example_param=1.0):  # only default arguments here
17 #Standard epy init stuff...
18         self.example_param = example_param
19         self.myacc=adi.adxl345(uri="local:")
20
21     def work(self, input_items, output_items):
22         """example: multiply with constant"""
23         output_items[0][:] =  input_items[0] * int(self.myacc.accel_z.raw)
24         return len(output_items[0])
25
```

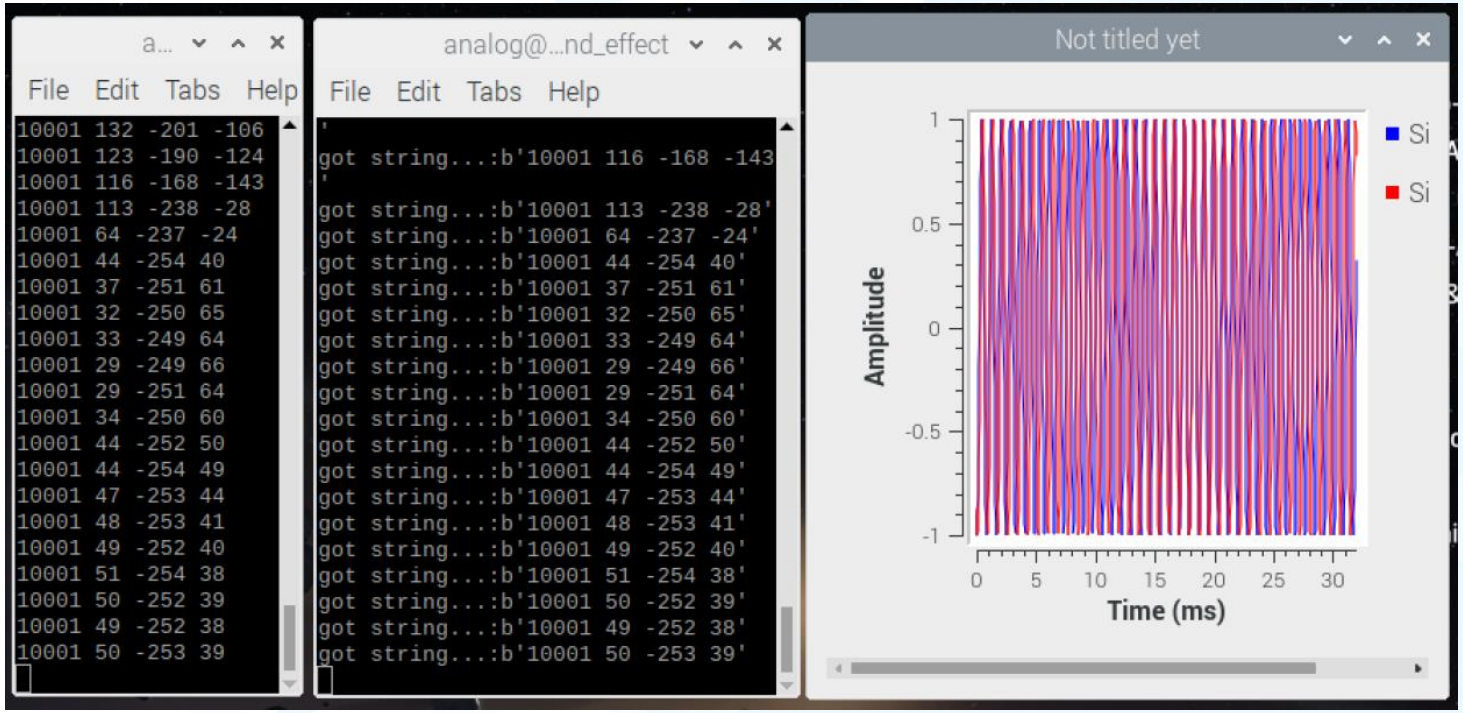
First attempt failed badly on GR3.7

- Some pyadi-iio functions worked under Python 2.7 by luck
  - (Not the ADXL345 ☹)
- Still possible to use “raw” libiio functions… but pyadi-iio has made the author lazy.

```
 9  import numpy as np
10  from gnuradio import gr
11  import adi
12
13  class blk(gr.sync_block):  # other base classes are basic_block, decim_bloc
14      """Embedded Python Block example - a simple multiply const"""
15
16      def __init__(self, example_param=1.0):  # only default arguments here
17  #Standard epy init stuff...
18          self.example_param = example_param
19          self.myacc=adi.adxl345(uri="local:")
20
21      def work(self, input_items, output_items):
22          """example: multiply with constant"""
23          output_items[0][:] =  input_items[0] * int(self.myacc.accel_z.raw)
24          return len(output_items[0])
25
```

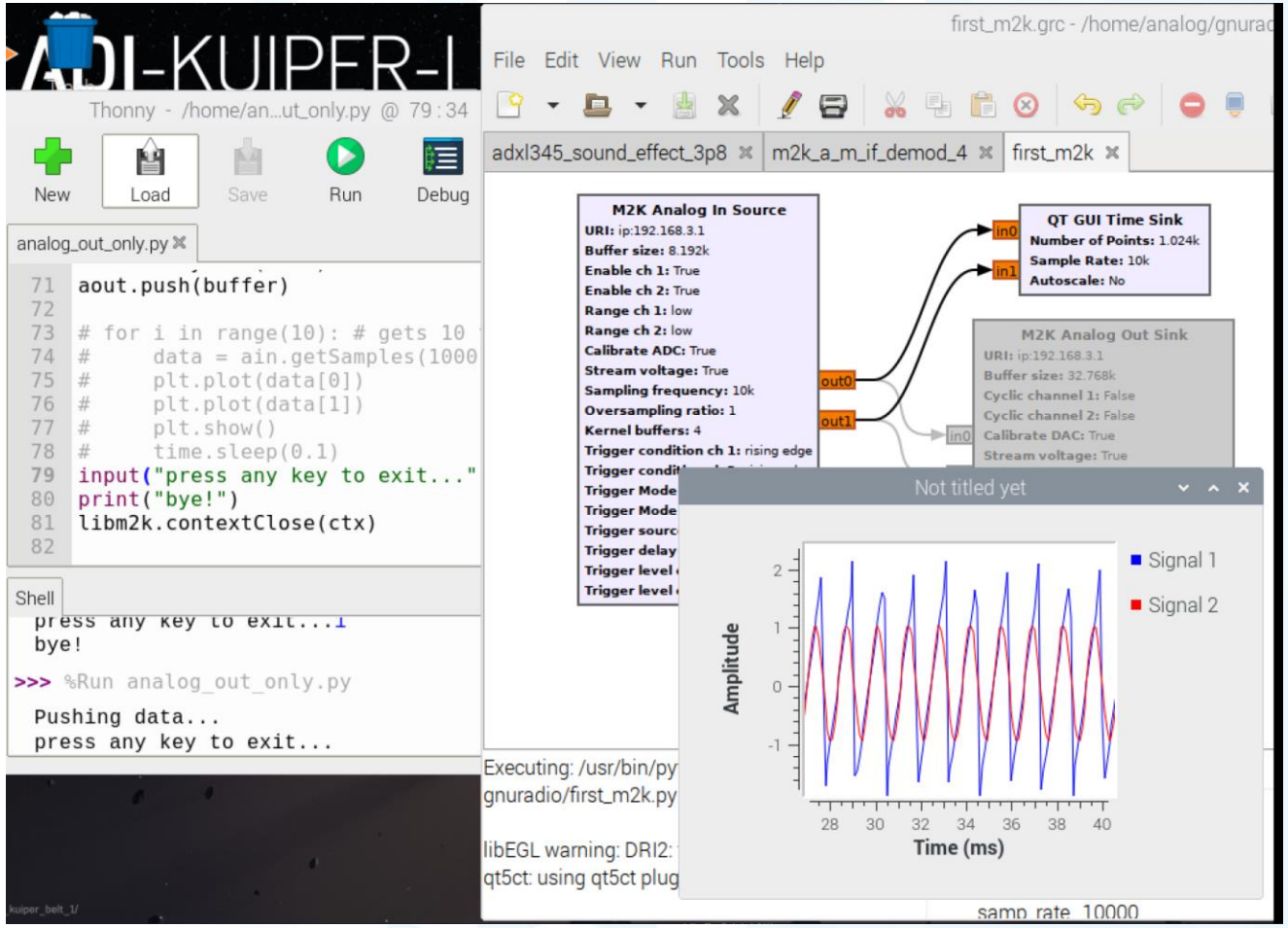
Final incarnation – external zmq publisher

- Separate script running in another terminal – pyadi-iio reads adxl345, publish to zmq
- Embedded Python block receives data, passes to flowgraph
- May be possible to use GR ZMQ blocks, but this method is infinitely flexible.
- Use your favorite environment for GUIs, other stuff you'd rather not do in GNURadio.



What else can we use external Python scripts for?

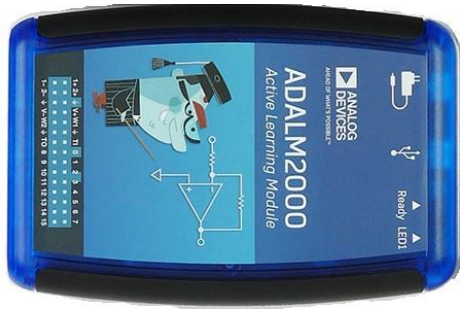
- Adalm2000 block!
- See “Build a Radio with an m2k and spare parts”
- How far can we milk the m2k in a radio application?
- Inspired by Jon Kraft’s A.M. to F.M. translator



Conceptual A.M. to F.M. translator

530-1600kHz A.M. broadcast band in

10Msps adequate to digitize entire A.M. Band (m2k will do 100Msps)



GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

Demodulate 455kHz A.M. I.F. Remodulate 200kHz wide F.M.

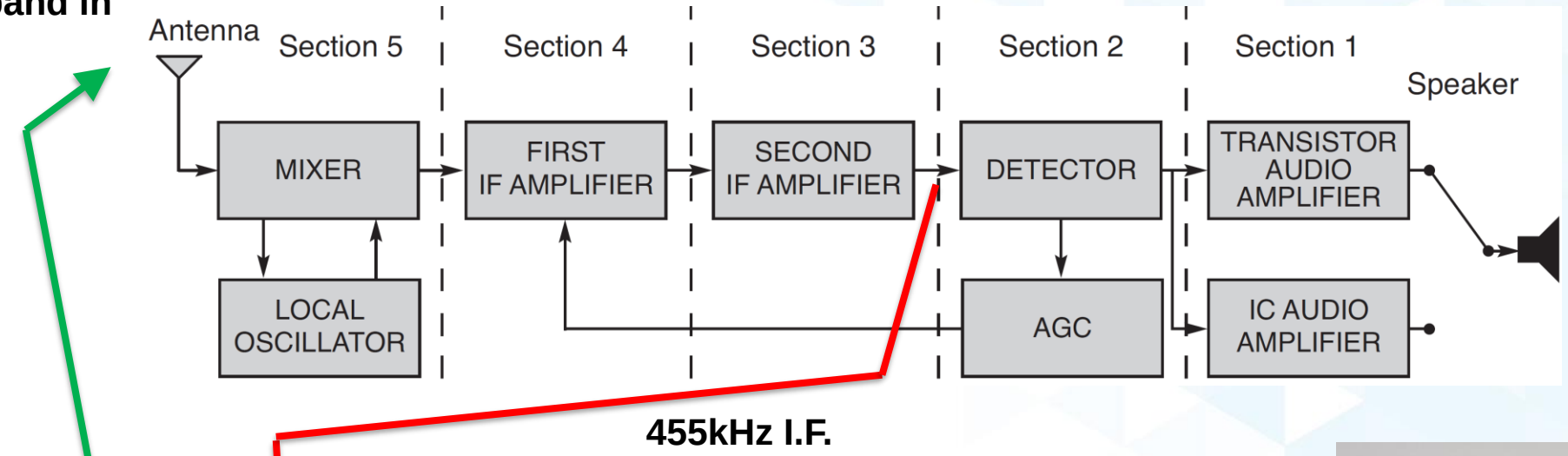
88-108MHz F.M. broadcast band out

Show Stopper: ~2Msps USB Bandwidth

Compromise: I.F. sampling

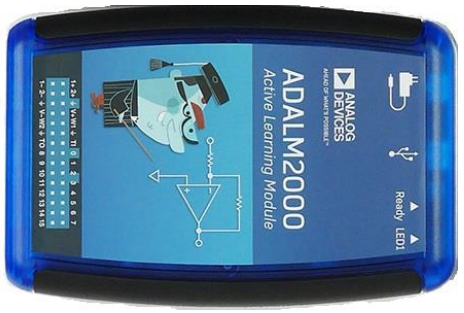
530-1600kHz A.M. broadcast band in

Educational A.M. Radio Kit



455kHz I.F.

88-108MHz F.M. broadcast band out

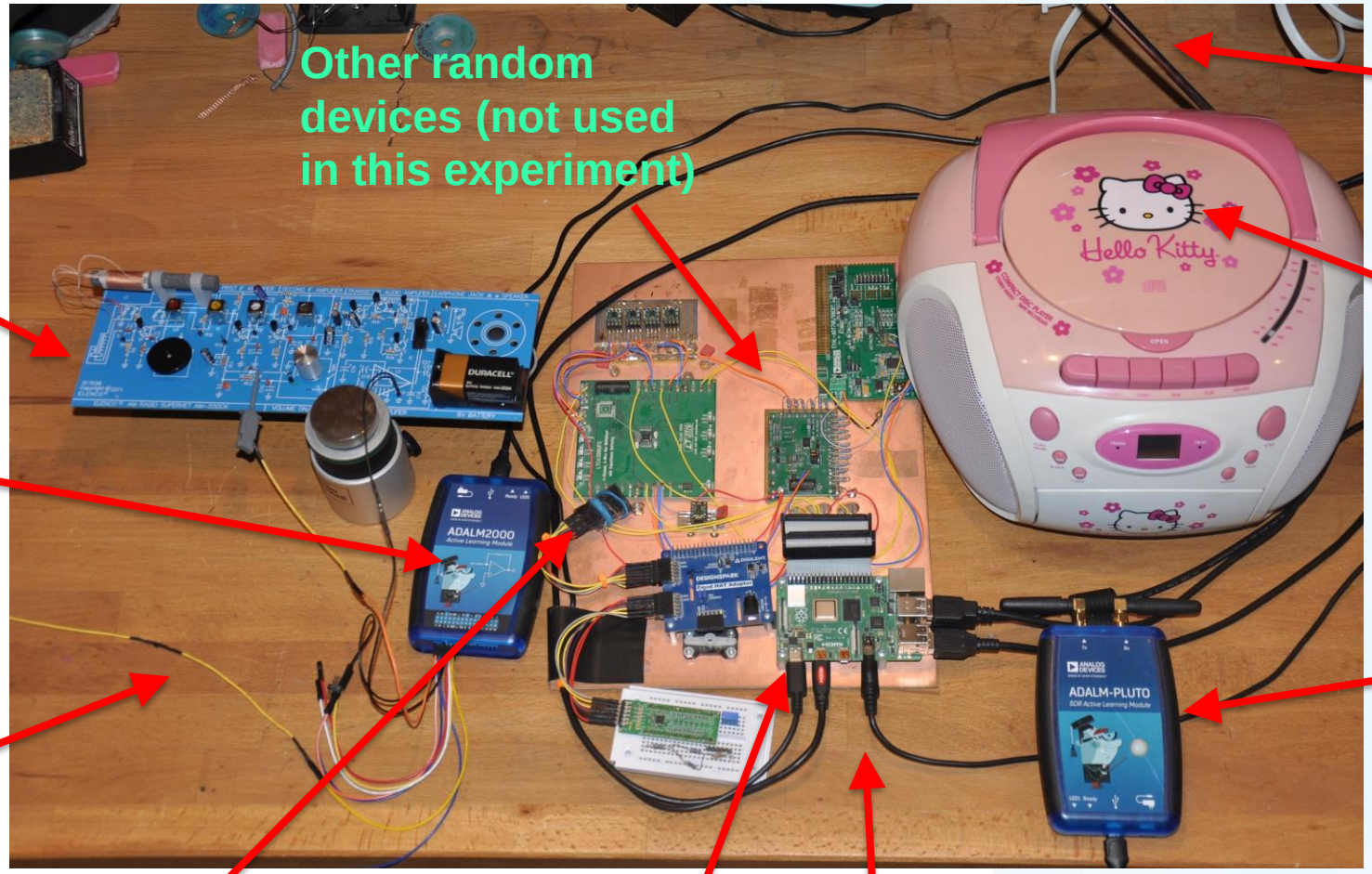


GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

Demodulate 455kHz A.M. I.F. Remodulate 200kHz wide F.M.

Where Python comes in: Generating A.M. R.F. test signals (@7.5Msps)

A.M. to F.M. Translator Overview



Elenco AM550 (A.M. Tune, downconvert to 455kHz)

ADALM2000: Digitize I.F. @2Msps (in GNURadio) Generate modulated RF test signal, 1MHz carrier (in Python)

R.F. Antenna

ADXL345 from previous experiment

Raspberry Pi 4

Audio Sink Output

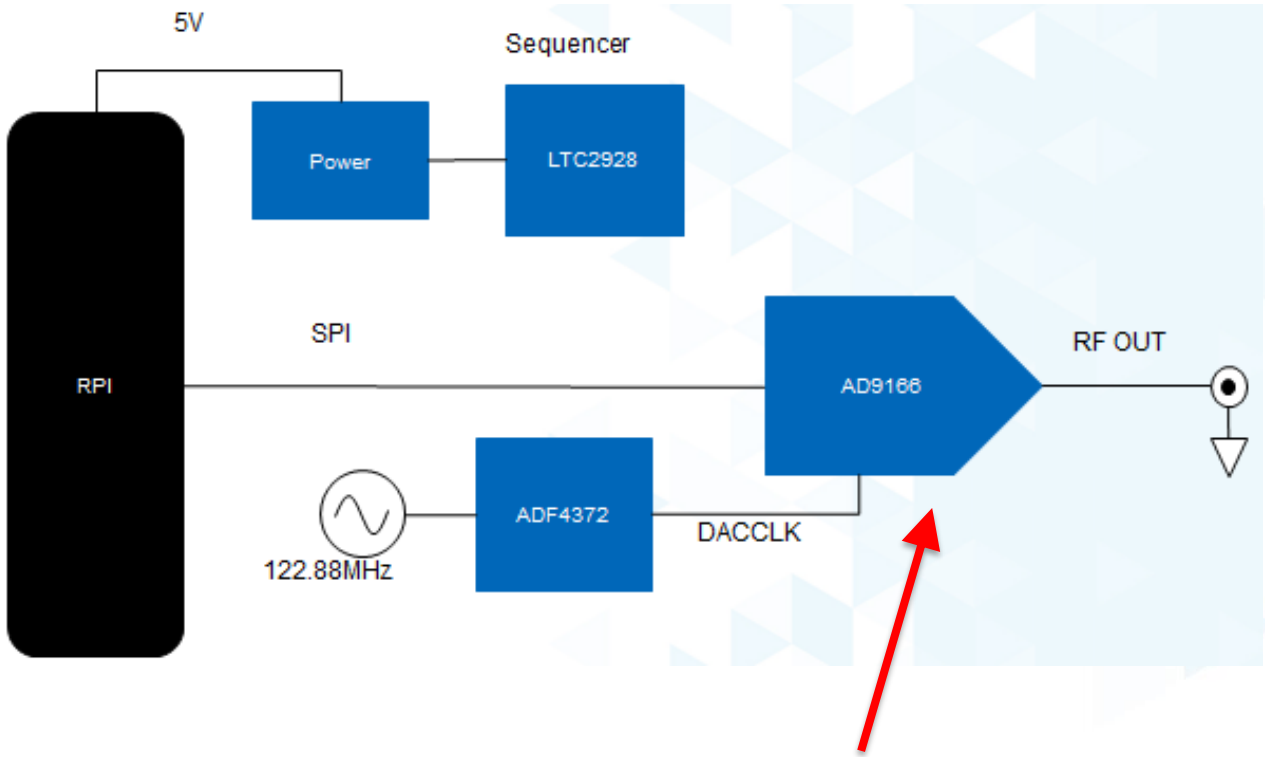
F.M. antenna

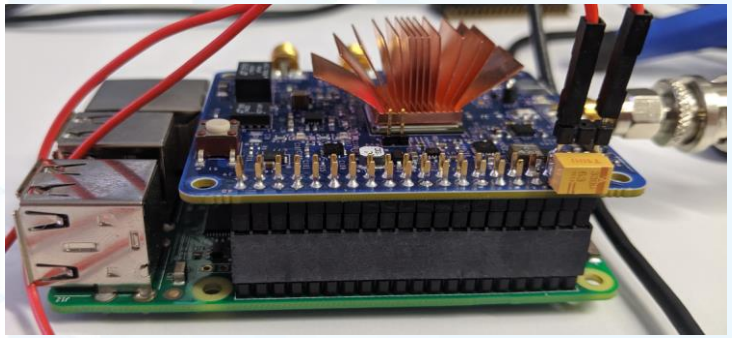
Hello Kitty (F.M. Rx)

ADALM-PLUTO (F.M. Tx)

Back to reality… practical stuff

CN0511 (In Development) D.S. to 5GHz Raspberry Pi-based Signal Generator





Production test jig = Rpi + Pluto + Pyadi-iio + Python script

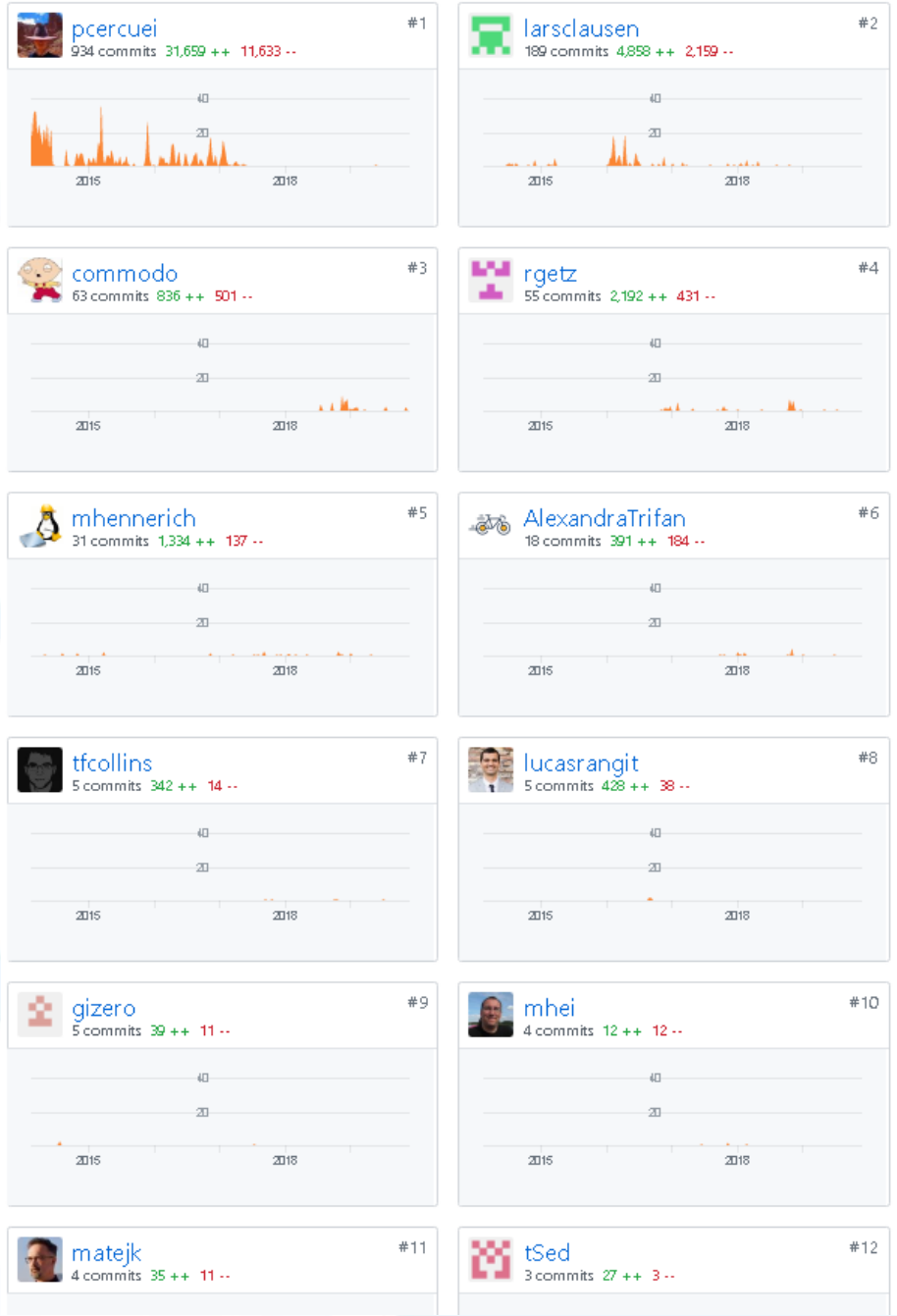
AD9166 DAC w/ integrated NCOs libiio, pyadi-iio compatible

Do-List

- **Python GUI / ZMQ publisher for phased array project**
- **Script flowgraph data processing / capture from external Python script**

Acknowledgements

- https://github.com/analogdevicesinc/libiio/graphs/contributors
- https://github.com/analogdevicesinc/pyadi-iio/graphs/contributors
- https://github.com/analogdevicesinc/adi-kuiper-gen/graphs/contributors
- Travis Collins – pyadi-iio
- Adrian Suciu and team – libm2k
- Mircea Caprioru – ADI Kuiper Linux



Thanks!

AHEAD OF WHAT'S POSSIBLE™