A Flexible Architecture for Monitoring Public Safety Communications using GNU Radio, RFNoC, and Python

Aaron Rossetto

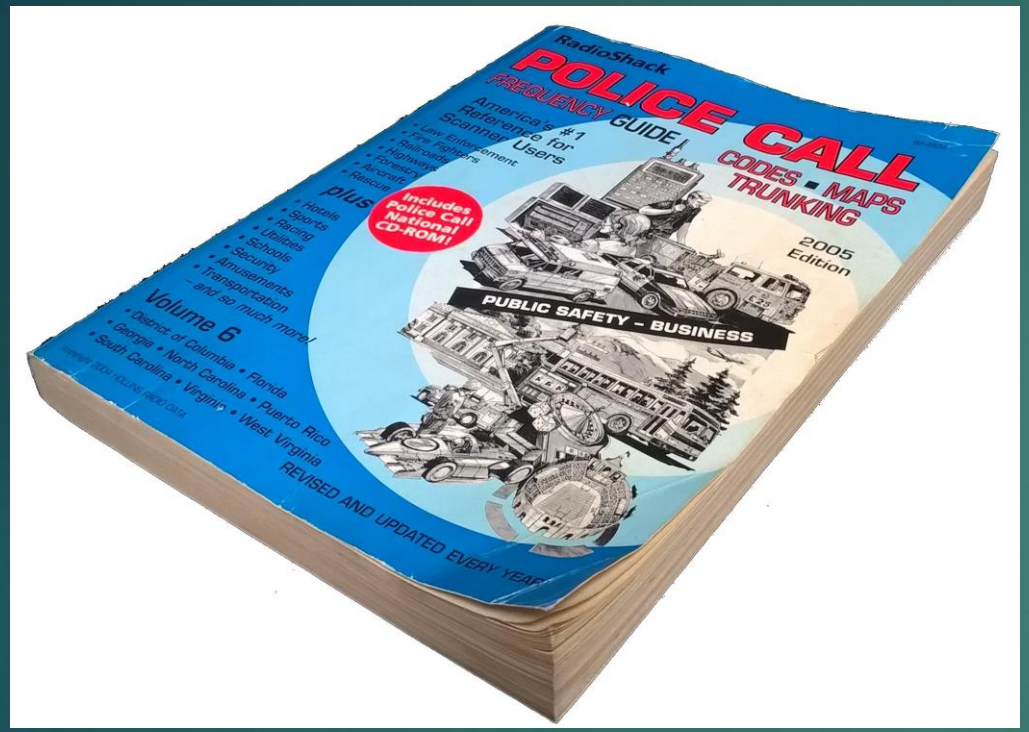
About Me

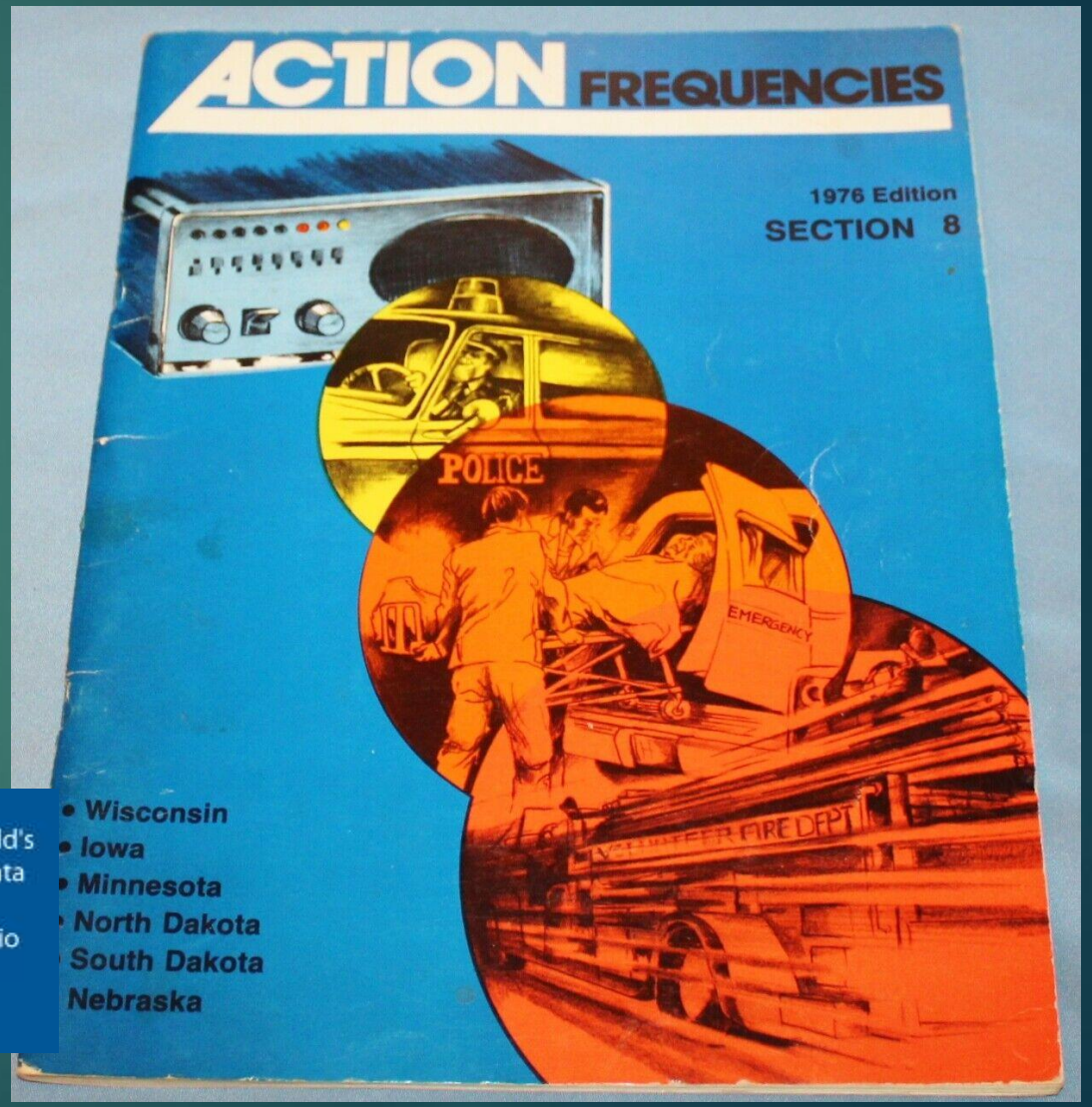
- Former NI/Ettus employee and UHD maintainer
- GRCon presentations
  - Exploring RFNoC with the UHD Python API
  - Meet the Family: RFNoC Blocks in UHD
  - Amateur Radio Meetup: A Look at Project 25 Digital Radio
- Long-time SDR and public safety monitoring enthusiast

Public Safety Monitoring

Listening to the radio communications of first responders, local, state, and federal agencies, and other governmental and community organizations as they respond to emergent situations, and the equipment and techniques used to do so

Public Safety Monitoring





Scanners


Hear What's Happening from the Comfort of Home with a New-for-'78 Realistic Scanning Receiver
Deluxe 8-Channel Table-Top Design
NEW FOR 78
Available Oct. 30, 1977
U.S. PAT. NO. 3,794,925/3,895,303
2 Bands VHF-Hi VHF-Lo 119.95
4 Bands VHF-Hi/Lo UHF-Hi/Lo 139.95
U.S. PAT. NO. 3,794,925
Available Oct. 30, 1977
• Built-In Scan Delay • Automatic and Manual Scanning
• Large Up-Front Speaker • High Quality IC Audio Output


VHF HI-LO
VHF/UHF

Scanners


REALISTIC
DIRECT ENTRY PROGRAMMABLE
AM/FM SCANNING RECEIVER
300 CHANNEL
PRO-2004
VOLUME
SQUELCH
SOUND
SQUELCH
DIMMER
MANUAL
PRI
LIMIT
SCAN
SPEED
DELAY
MODE
LOCK-OUT
STEP
DIRECT
LOCK-OUT
REVIEW
RESET
MONITOR
PROGRAM
ENTER
CLEAR

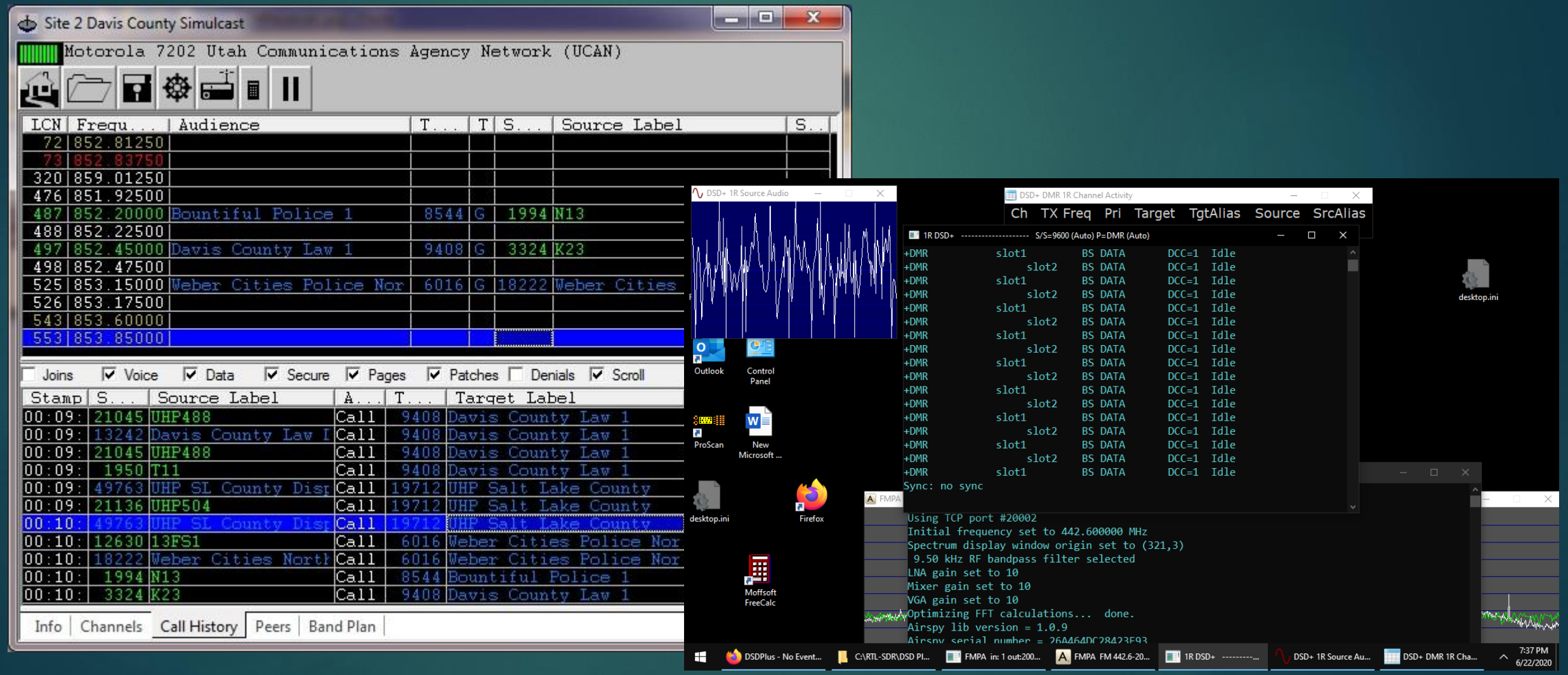

Uniden
TRUNKTRACKER III
SCAN
MANUAL
SRC
AUTO
LIMIT
PRI
TRUNK
DELAY
DATA
CTCSS/DCS
L/O
LINE
EXT SP

Scanners


Uniden
Squelch
Range
Atten
GPS
Help
Northeast Tarrant County
Grapevine
Police 1 (Dispatch)
Law Dispatch 12848
Menu
Replay
Record
Mute
HOMEPATROL-II
CHARGE


Uniden
SDS200
PUSH/FUNC
Fort Worth Regional Radio Sy...
Police
Tarrant County Sheriff (Laye...
Fort Worth Layer 2
Sheriff Patrol
GID:5834
Law Dispatch
SYSTEM
DEPT
CHANNEL

Computer-Based Solutions



Gaming Platform-Based Solutions

https://old.reddit.com/r/RTLSDR/comments/xf9v16/steam_deck_portable_trunking_setup_using_sdrtrunk/


Is this a scanner ?

Monitoring Option Deficiencies

Scanners

- Made for broad audiences and common use cases
- No or only limited customization possible
- Typically have a single tuner
- Inscrutable UI/UX design choices

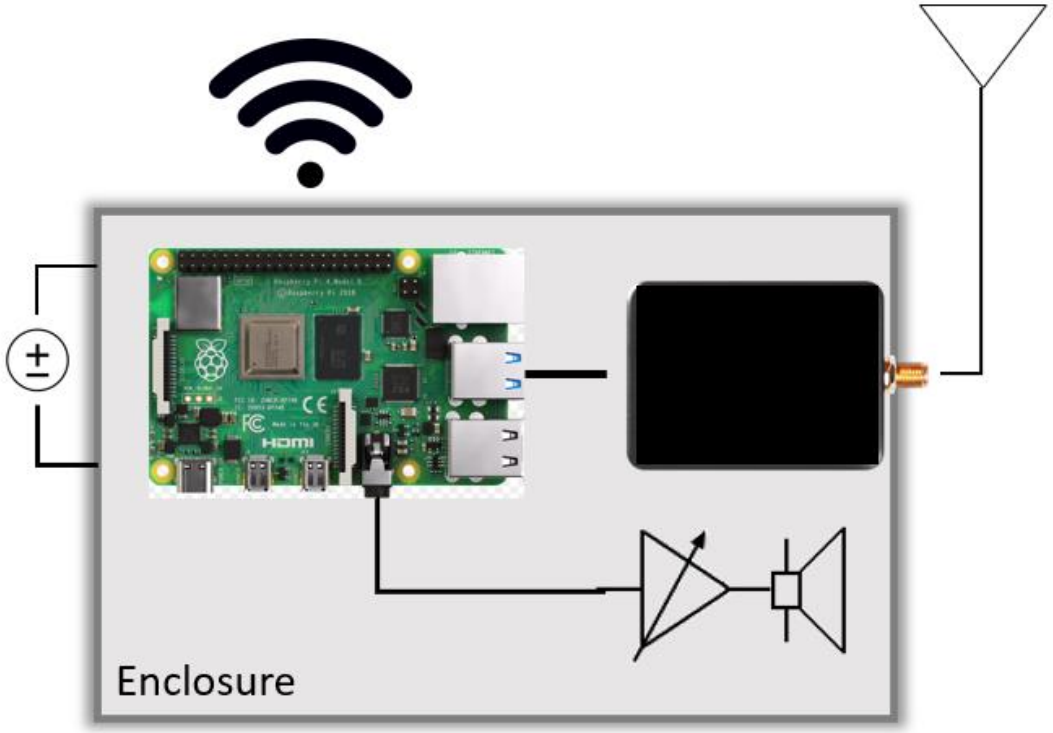
Computer-Based Scanners

- Often difficult for novices to use/configure
- Require dedicated compute resources
- Customization possible with specific knowledge and/or toolset
- Inscrutable UI/UX design choices

My dream

"DIY OSS Trunked Radio Scanner"

- RPi, SDR module, and audio amp/speaker in an enclosure
- Headless (web-based mobile friendly UI for interactive control)
- Wi-Fi enabled for updating, configuration, audio streaming, and cloud audio backup
- GNU Radio support, of course :)
  - Maybe even implemented in GR



DIY Scanner Goals

- Flexibility above all
  - Front end flexibility (radio, channel selection, demodulation)
  - Back end flexibility (the 'business logic')
- Leverage modern SDR techniques/hardware and software packages
- Simplify back end development (the 'user experience' bits)
  - Decouple from and abstract away front end
  - Insulate implementer as much as possible from communications protocol
  - Accelerate development via interpreted language and a rich library ecosystem for building monitoring applications

Introducing gr-scanner!

A GNU Radio module to simplify the creation of customized listener-oriented monitoring solutions focusing on P25 trunked public safety radio systems

gr-scanner Overview

- Front end
  - GNU Radio flowgraph for signal acquisition, channel selection, 4FSK demodulation, and P25 message framing
  - Designed to support multiple simultaneous channel acquisition (e.g., a control channel and a separate traffic channel)
- Back end
  - Python module implementing the 'business logic' of the monitoring application
  - Designed to be decoupled from GNU Radio and front end flowgraph
  - Data exchange accomplished via *lingua franca* of JSON messages

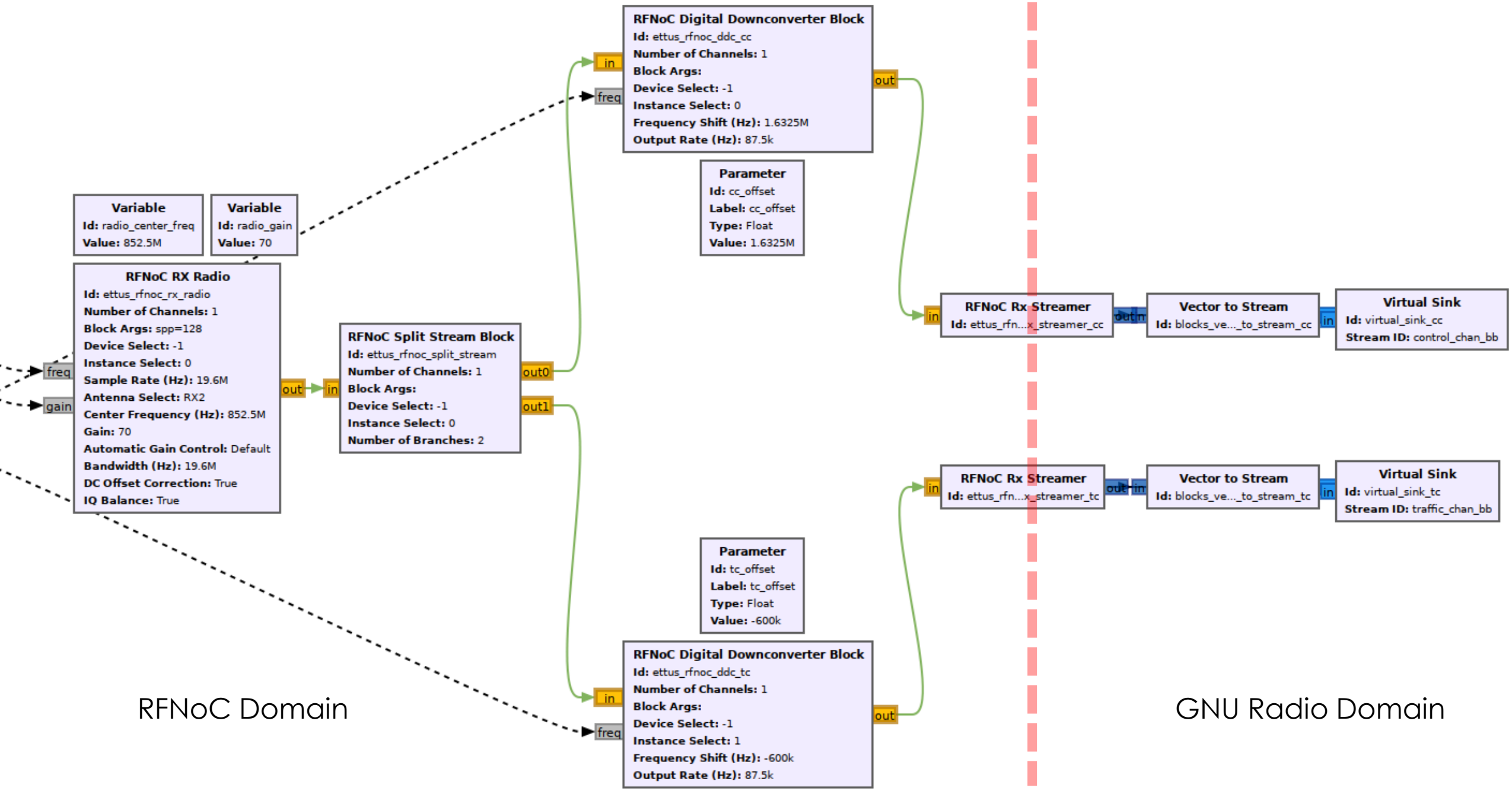
Credit Where Credit Is Due

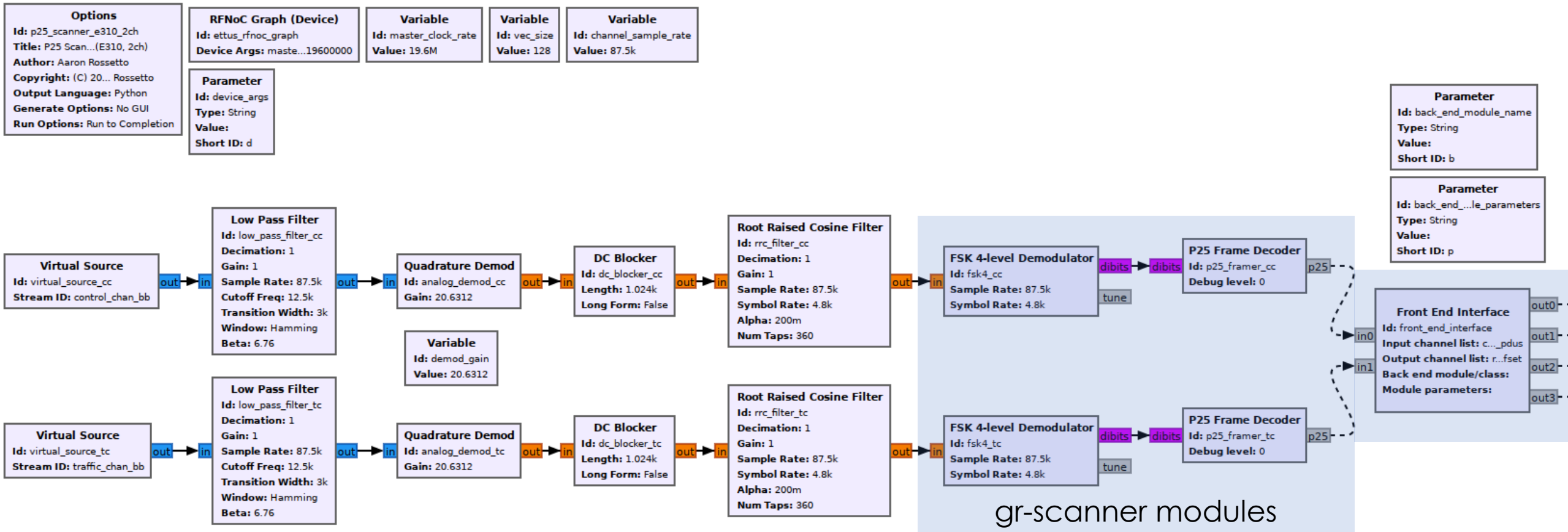
- Credit to OP25 project authors and contributors from which I based much of the P25 framer code
  - gr-op25 uses many similar techniques
  - Supports more than P25 phase 1

Max H. Parke, KA1RBI
Jonathan Naylor, G4KLX
Michael Ossmann

Pavel Yazev
Hard Consulting Corporation
<+YOU OR YOUR COMPANY+>

- And all the other contributors to GR, UHD, cmake, etc. 👍

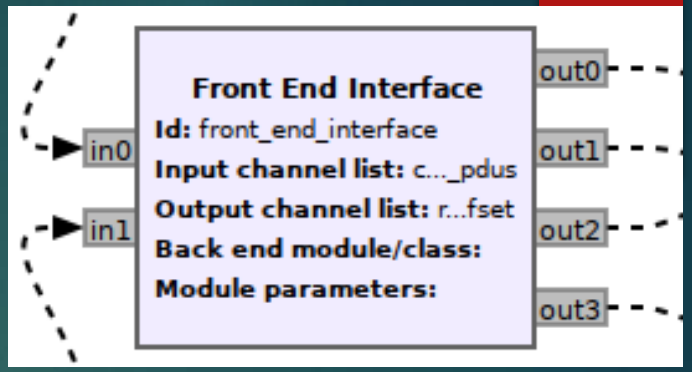

RFNoC Digital Downconverter Block
Id: ettus_rfnoc_ddc_cc
Number of Channels: 1
Block Args:
Device Select: -1
Instance Select: 0
Frequency Shift (Hz): 1.6325M
Output Rate (Hz): 87.5k
in
freq
out
Parameter
Id: cc_offset
Label: cc_offset
Type: Float
Value: 1.6325M
Variable
Id: radio_center_freq
Value: 852.5M
Variable
Id: radio_gain
Value: 70
RFNoC RX Radio
Id: ettus_rfnoc_rx_radio
Number of Channels: 1
Block Args: spp=128
Device Select: -1
Instance Select: 0
Sample Rate (Hz): 19.6M
Antenna Select: RX2
Center Frequency (Hz): 852.5M
Gain: 70
Automatic Gain Control: Default
Bandwidth (Hz): 19.6M
DC Offset Correction: True
IQ Balance: True
freq
gain
out
RFNoC Split Stream Block
Id: ettus_rfnoc_split_stream
Number of Channels: 1
Block Args:
Device Select: -1
Instance Select: 0
Number of Branches: 2
in
out0
out1
RFNoC Rx Streamer
Id: ettus_rfn...x_streamer_cc
Vector to Stream
Id: blocks_ve..._to_stream_cc
Virtual Sink
Id: virtual_sink_cc
Stream ID: control_chan_bb
RFNoC Rx Streamer
Id: ettus_rfn...x_streamer_tc
Vector to Stream
Id: blocks_ve..._to_stream_tc
Virtual Sink
Id: virtual_sink_tc
Stream ID: traffic_chan_bb
Parameter
Id: tc_offset
Label: tc_offset
Type: Float
Value: -600k
RFNoC Digital Downconverter Block
Id: ettus_rfnoc_ddc_tc
Number of Channels: 1
Block Args:
Device Select: -1
Instance Select: 1
Frequency Shift (Hz): -600k
Output Rate (Hz): 87.5k
RFNoC Domain
GNU Radio Domain

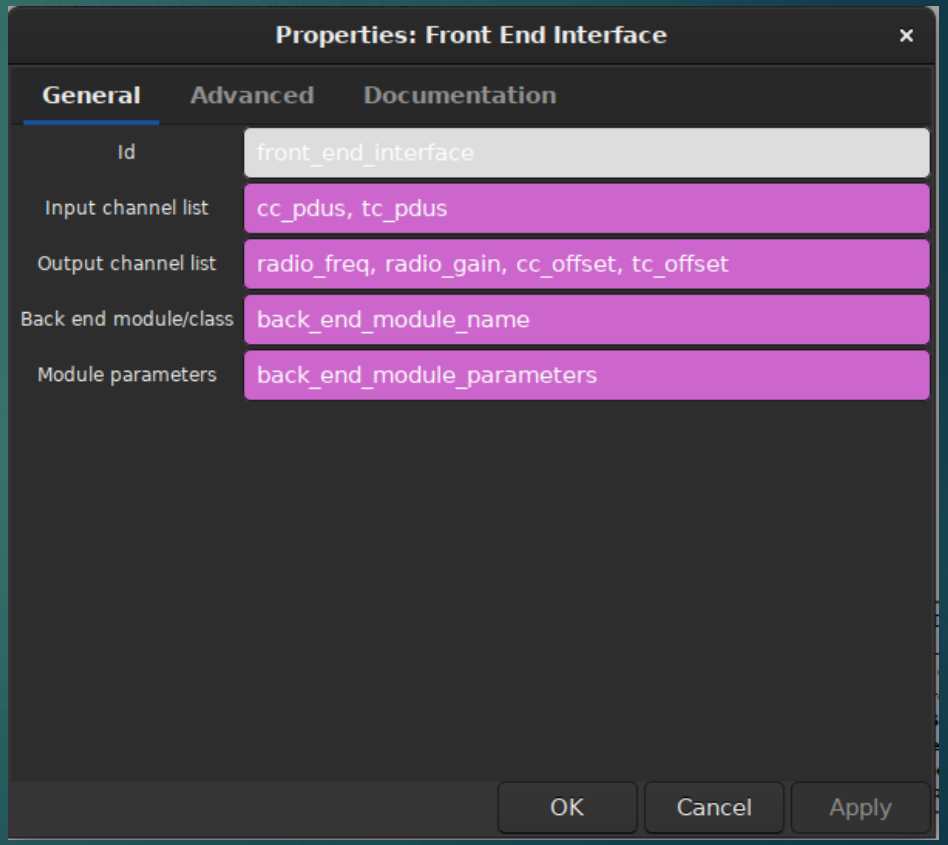


GNU Radio Domain

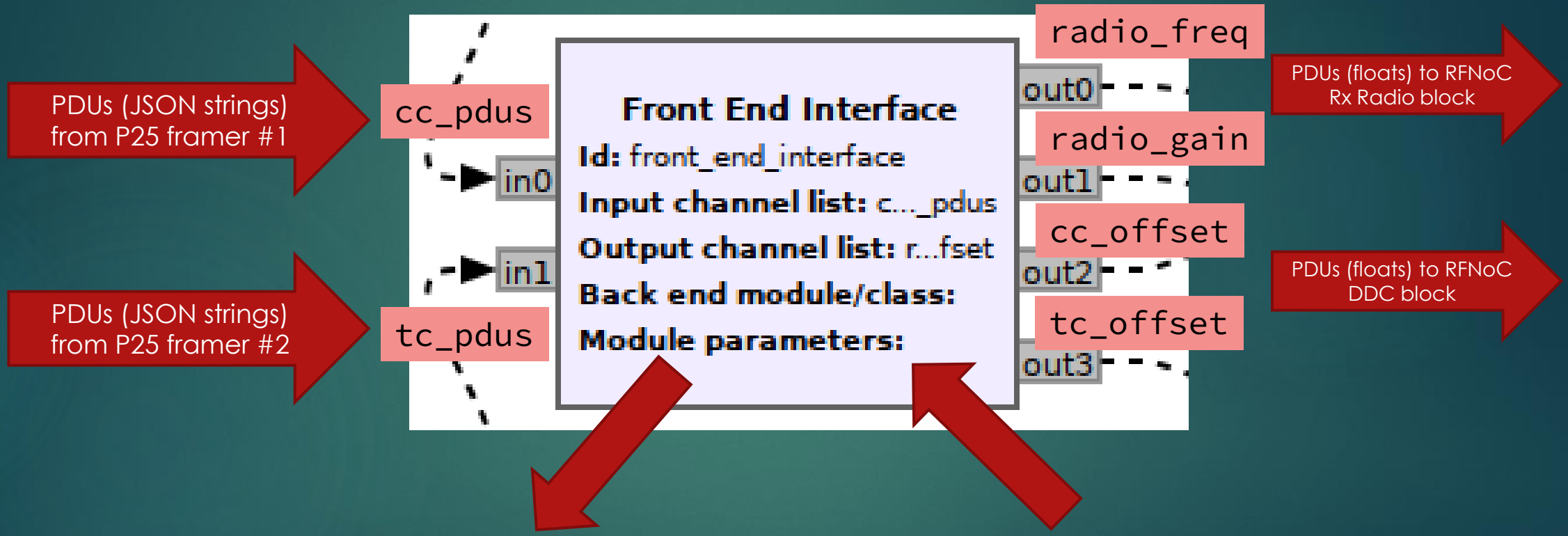
Front End Interface

- GNU Radio Python block connecting front and back ends
  - Loads Python module and instantiates named class with given parameters
  - Accepts PDUs from input ports and proxies to `receive_pdu()` on class
  - Outputs PDUs on output ports from back end via `send_pdu()` method
  - Maps in*N* and out*N* ports to names provided in input and output channel lists





Front End Interface in P25 Scanner Front End

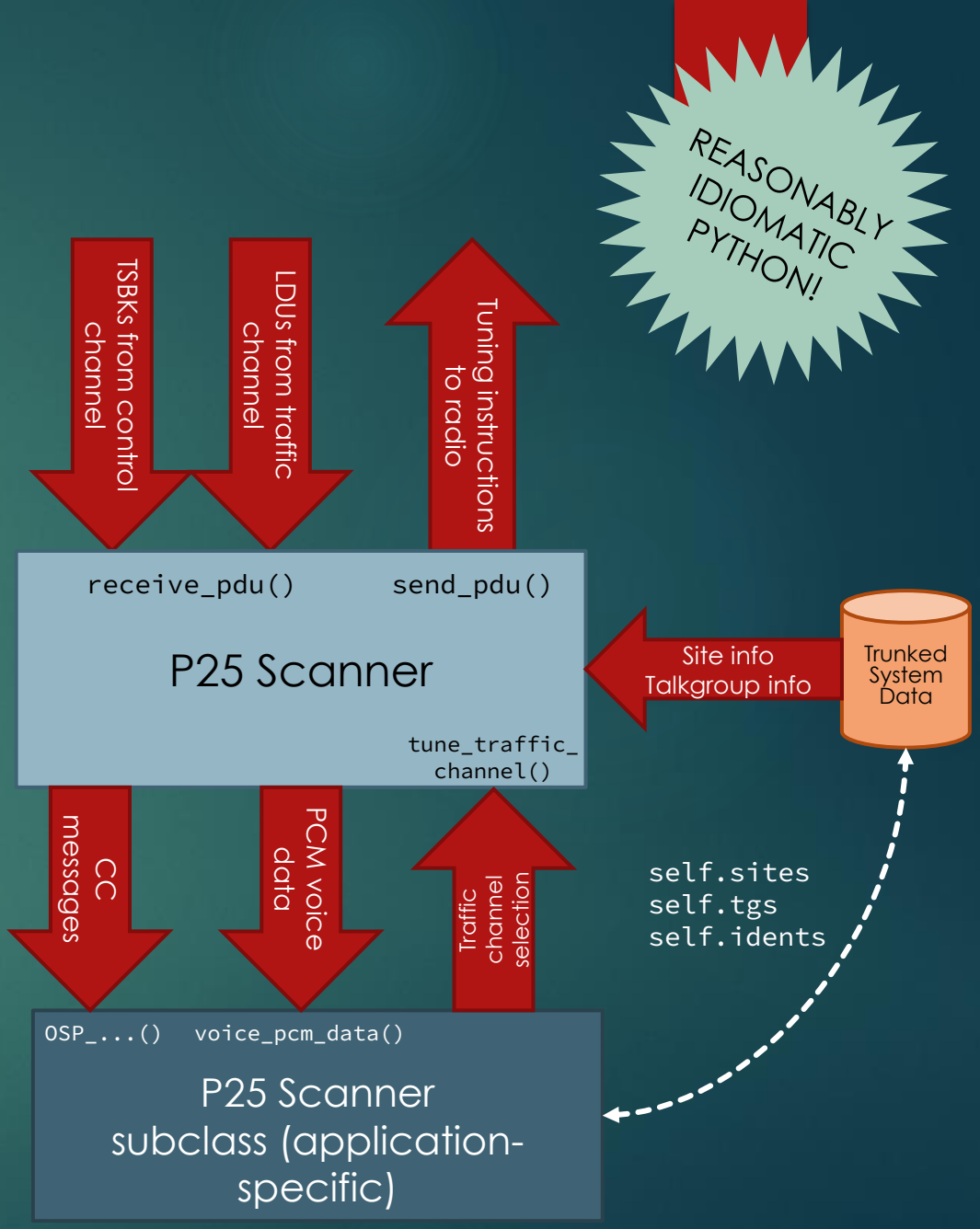



```
back_end_class.receive_pdu('cc_pdus',
{'p25_du': {'nac': '0x137', 'duid': '0x7',
'tsbk': '90005235FAE30A5761E1C01264',
'ok': 1}}
```

```
send_pdu_fn('tc_offset',
{'type': 'float',
'value': '-161500'})
```

Back End: P25 Scanner

- Python module; no GR dependencies
- Base class handling common P25 trunked system decoding tasks
  - Configures front end radio for trunked system reception
  - Interprets trunked control channel messages and calls user-defined functions
  - Decodes digital voice packets on traffic channels to PCM data
  - Parses trunked system data files from Radio Reference database and provides access via dictionaries
- Intended to be subclassed to implement application-specific behaviors



Back End: P25 Decoder Ring

- Map of TSBK messages to field names and decoders
  - List of byte offset, masks, and shifts for each field, or lambda with decoding code
- Method with same name as message called on subclass with fields as parameters

```
def OSP_GRP_V_CHANNEL_GRANT(self,
service_opts, freq, group,
source):
  ...
```

```
# Master list of TSBK OSPs
class tsbk_osps(Enum):
    # Standard OSPs (vendor ID = 0x00)
    OSP_GRP_V_CHANNEL_GRANT = (0, 0x00)
```

```
# Describe decoding details for each OSP
#   Key is the tsbk_osps enumeration name (string)
#   Value is a dictionary with key being the parameter name and the value
#   being how to calculate the value for the parameter
tsbk_osp_decoders = {
    'OSP_GRP_V_CHANNEL_GRANT': {
        'service_opts': [(2, 0xff, 0)],
        # If the key is a callable, it is a lambda taking the buffer of TSBK
        # payload bytes, returning the value associated with the parameter
        'freq': lambda self, data:
            self.system.channel_to_freq((data[3] << 8) | data[4]),
        # If the key is a list, it is a list of byte offsets, masks, and shifts
        # (positives are right-shifts, negatives are left-shifts) to apply,
        # with the results being bitwise ORed together
        'group': [(5, 0xff, 0), (6, 0xff, 0)],
        'source': [(7, 0xff, 0), (8, 0xff, 0), (9, 0xff, 0)]
    },
    'OSP_GRP_V_CHANNEL_GRANT_UPDATE': {
        'freq0': lambda self, data:
            self.system.channel_to_freq((data[2] << 8) | data[3]),
        'group0': [(4, 0xff, 0), (5, 0xff, 0)],
        'freq1': lambda self, data:
            self.system.channel_to_freq((data[6] << 8) | data[7]),
        'group1': [(8, 0xff, 0), (9, 0xff, 0)],
    },
```

Back End: IMBE Audio Decoder

- LDU1/2 P25 DUs on traffic channel port sent to IMBE decoder
  - Fixed point implementation by Pavel Yazev
  - Decoder is standalone shared library built alongside gr-scanner
- Calls subclass `voice_pcm_data()` with raw 16-bit PCM data at 8 kHz

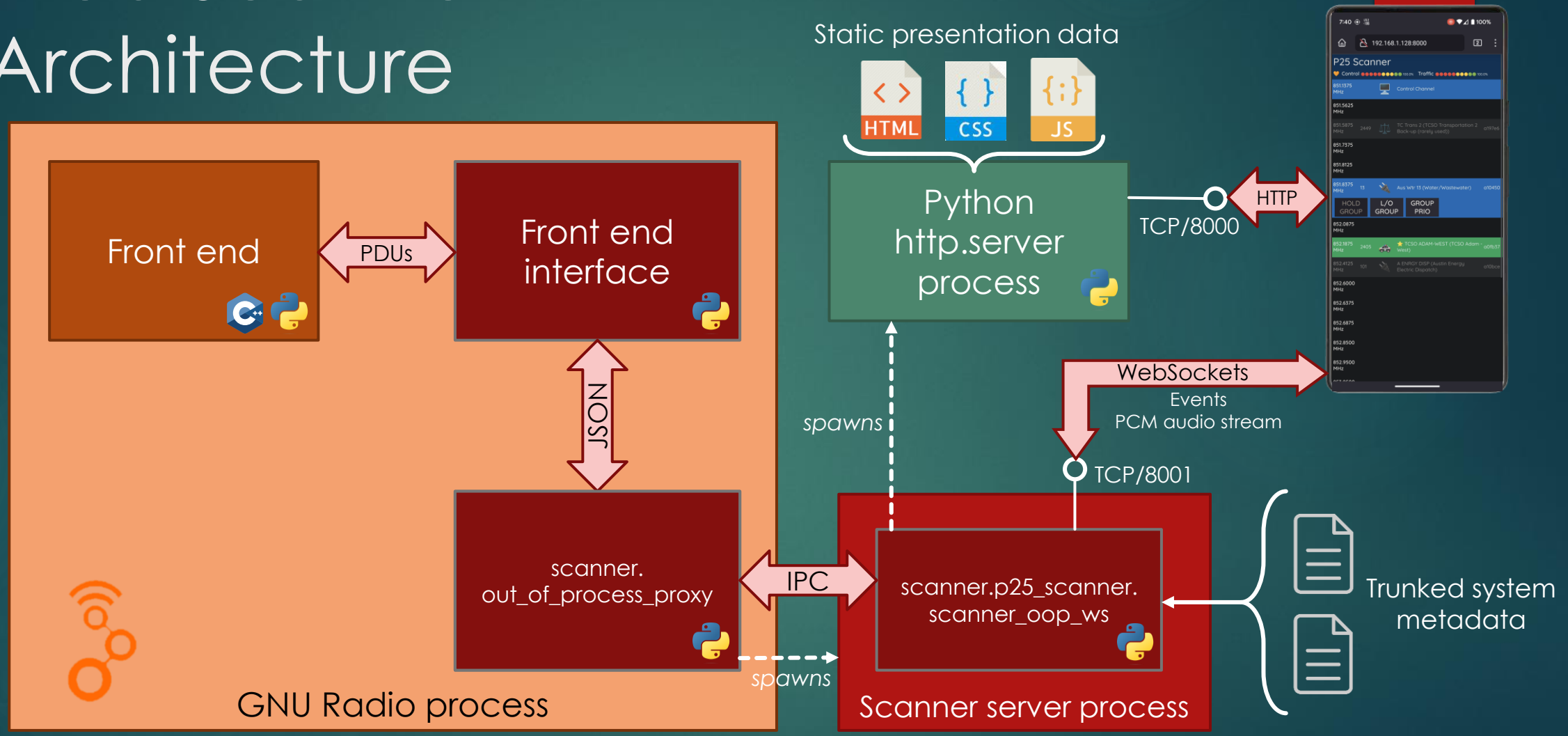
Back End: Out Of Process Proxy

- Spawns child process and loads specified module and instantiates named class
  - Creates pipes for IPC with proxy
  - Provides class with port names and input/output pipe handles
  - Calls class `main_loop()`
- Serializes `receive_pdu()` and `send_pdu()` calls via pipes
- Isolates back end from GR Python process

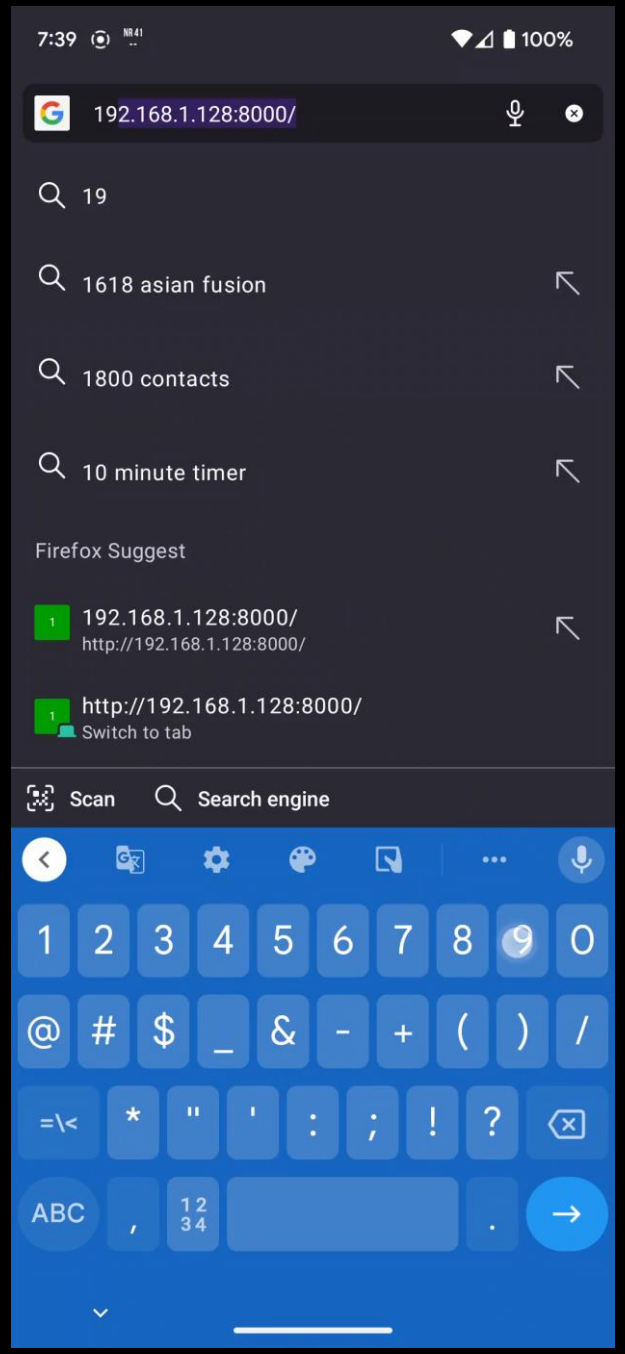
Example: Simple Web Scanner

- PoC of mobile-oriented web-based P25 scanner written in Python
  - View real-time site activity with users
  - Monitor audio
  - Lock out or prioritize specific talkgroups
  - Visual feedback of signal quality
- Playground to try out UI/UX ideas and refine framework

Web Scanner Architecture

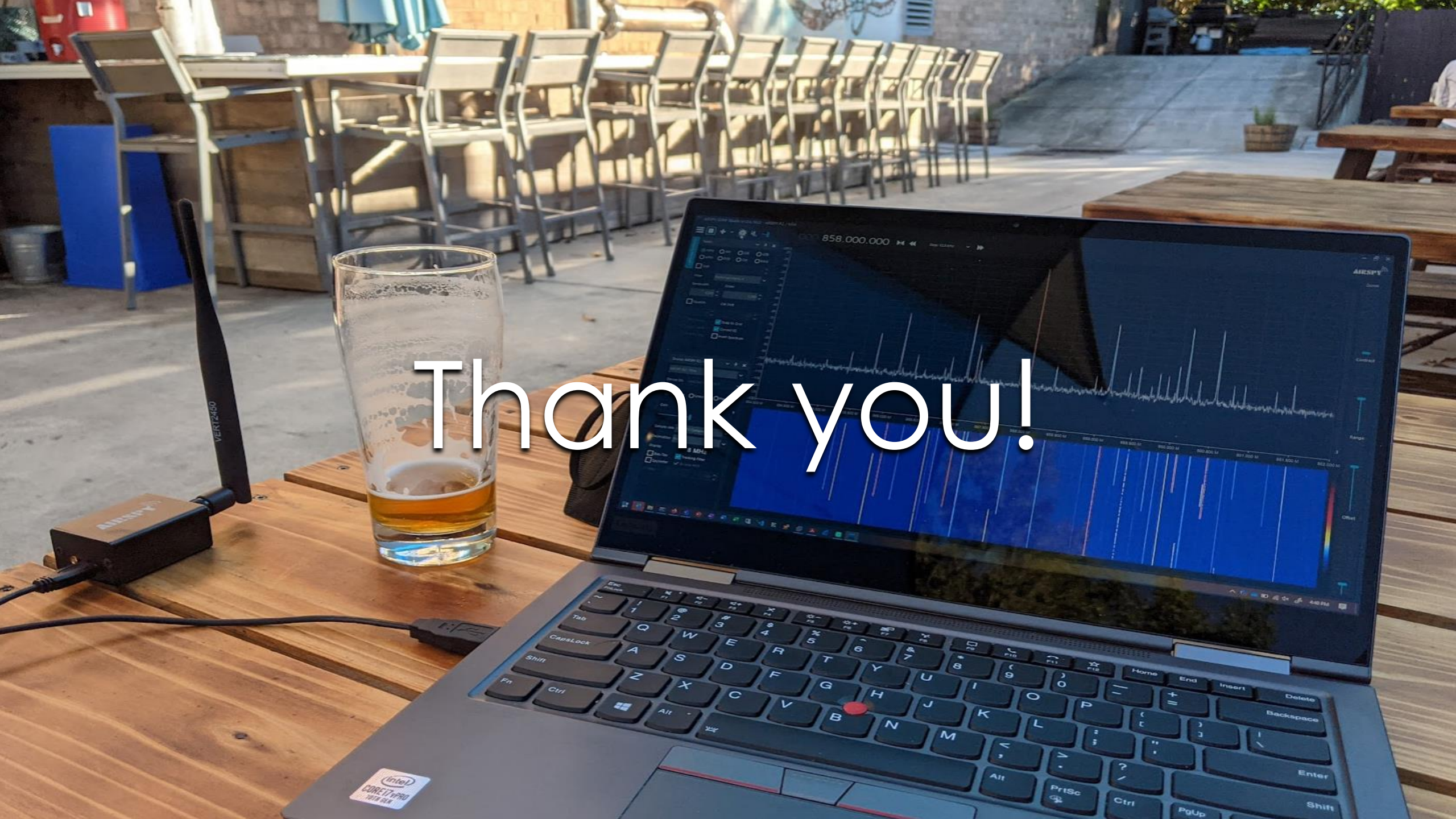


Demo


7:39
100%
192.168.1.128:8000/
19
1618 asian fusion
1800 contacts
10 minute timer
Firefox Suggest
192.168.1.128:8000/
http://192.168.1.128:8000/
http://192.168.1.128:8000/
Switch to tab
Scan
Search engine
1 2 3 4 5 6 7 8 9 0
@ # $ _ & - + ( ) /
=\< * " ' : ; ! ?
ABC , 12 34 .

How To Play Along

- https://github.com/meowdul8/gr-scanner
  - Fork it, improve it, send me your PRs!
  - Do cool things!
- Potential areas of improvement/feature additions:
  - TLC for the overall repo
  - Flowgraphs for other SDRs out of the box
  - Support for other trunked system types
  - Examples for different applications
  - Laugh at, then improve, my Javascript code
  - etc. etc. etc.


Thank you!