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

Welcome to Radio Reference

RadioReference.com is the world's largest radio communications data provider, featuring a complete frequency database, trunked radio system information, and FCC license data.
Scanners

Hear What’s Happening from the Comfort of Home with a New-for-’78 Realistic Scanning Receiver

Deluxe 8-Channel Table-Top Design

Available Oct. 30, 1977

2 Bands
VHF-Hi/Lo

119.95

4 Bands
VHF-Hi/Lo
UHF-Hi/Lo

139.95

- Built-In Scan Delay
- Automatic and Manual Scanning
- Large Up-Front Speaker
- High-Quality IC Audio Output

Available Oct. 30, 1977

U.S. PAT. NO. 3,794,925/3,805,533

U.S. PAT. NO. 3,794,925

Available Oct. 30, 1977

VHF/UHF
Scanners
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. 👍
GNU Radio Domain

gr-scanner modules
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 `inN` and `outN` ports to names provided in input and output channel lists
Front End Interface in P25 Scanner
Front End

```
back_end_class.receive_pdu('cc_pdu',
    {'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

P25 Scanner

- Receive PDU
- Send PDU
- Tune traffic channel
- Trunked System Data
  - Site info
  - Talkgroup info
- P25 Scanner subclass (application-specific)
  - OSP_...()
  - Voice pcm data()
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):
    ...

```python
# Master list of TSBK OSPs
class tsbk_osps(Enum):
    # Standard OSPs (vendor ID = 0x00)
    OSP_GRP_V_CHANNEL_GRANT = (9, 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 list, 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': [(6, 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': [(6, 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 \texttt{voice_pcm_data()} with raw 16-bit PCM data at 8 kHz
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

- Front end
- Front end interface
- GNU Radio process
- Scanner server process
- Scanner server process spawns
- Scanner server process spawns
- Scanner server process IPC
- Scanner server process spawns
- Python http.server process
- Python http.server process
- HTTP
- TCP/8000
- TCP/8001
- WebSockets
- Events
- PCM audio stream
- Trunked system metadata
- Static presentation data
- HTML
- CSS
- JS
- scanner.p25_scanner
- scanner_oop_ws
- scanner_out_of_process_proxy
- PDUs
Demo
How To Play Along

- [https://github.com/meowdul8/gr-scanner](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!