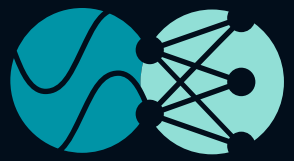


FunKI

Getting started with GNUradio

Workshop, 22.09.2021

creonic ip cores & system solutions

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

intel

MOTIUS WE R&D.

NOKIA

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Universität Bremen

University of Stuttgart Germany

GEFÖRDERT VOM

Bundesministerium für Bildung und Forschung

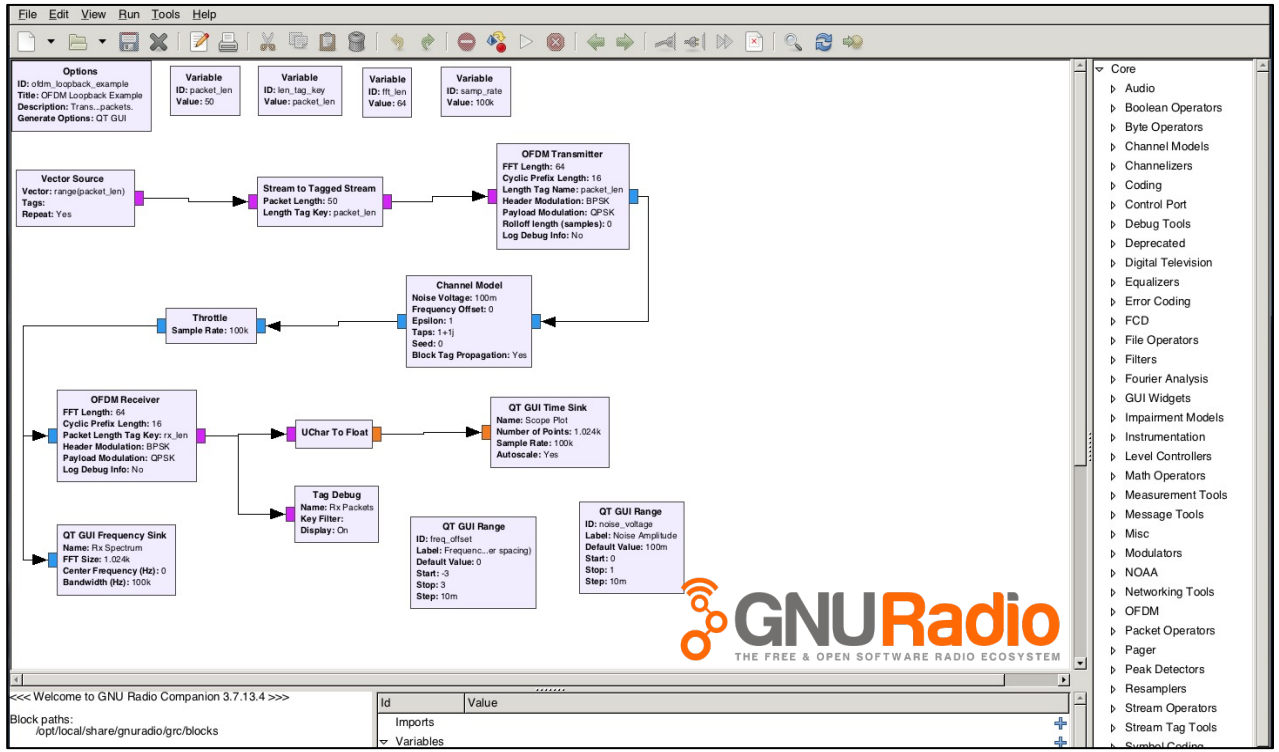
Agenda

1. Prerequisites
    1. Overview, history and future of GNUradio
    2. Installation
2. Basic concetps
    1. First start
    2. Overview of the GNUradio-companion tool
    3. Blocks and the data-flow structure
3. Our first flow graph
    1. Interacting with audio – source and sink blocks
    2. Running the graph – understanding the ”compilation-process”
    3. Threshold-blocks and when to use them
    4. Running a graph via Terminal
    5. File-sources, selectors and other concepts
4. Hierarchical blocks
5. Embeded python blocks, the quick way to incorperate custom functionality
6. Preview:
    1. Creating you own custom blocks and out of three modules (grmodtool)
    2. Interfacing with hardware

1: Prerequisites

Overview GNURadio

- GNU radio is a free, powerful and **open-source** SDK for signal processing
- Can either be used for simulation or to interface real world hardware
- Broad **compatibility** with external RF-Hardware
- Written in C++ and Python



History and state of GNURadio today

- GNUradio started as the brainchild of John Gilmore and Eric Bossom in 2001 and was based on Pspectra
- Went through several overhauls with it's more than 20 years history
- Most recent stable release today is 3.9.2
- Supported by many chairs, companies and enthusiast alike

Web presence	Docu	OOTs
Website	Wiki	Library of OOTs
Youtube	Offical Blog	Porting guide
Conference	BastiBl_Blog	
	GRC-list	

Original author(s)	Eric Blossom
Developer(s)	GNU Radio Community Project Lead: Ben Hilburn Maintainer: Marcus Müller
Initial release	2001; 20 years ago
Stable release	3.9.2.0[1] / 10 June 2021; 3 months ago
Repository	github.com/gnuradio/gnuradio.git
Written in	C++, Python
Operating system	Cross-platform
Available in	English
Type	Radio
License	2007: GPL-3.0-or-later[2] 2001: GPL-2.0-or-later[3]
Website	www.gnuradio.org

Installation GNURadio

- GNUradio officially supports all major operating systems including Windows, Mac and Linux
- In reality only Linux can be recommended
- From personal experience Ubuntu (e.g. LTS 20.04) is advised
- Apart from that, Kali Linux (preinstalled GRC) or a Virtual Machine can be recommended

Prebuilt Binaries (via package manager)	using pybombs	from source	Os with packed GNUradio	Using a VM

For Ubuntu:
`sudo apt install gnuradio`
For other distros
	Pybombs is a build system specficially for GNUradio it is as simple to use as:
	`sudo -H pip3 install PyBOMBS`
	`pybombs auto-config`
	`pybombs recipes add-defaults`
	`pybombs prefix init ~/gnuradio -R gnuradio-defaulty`
		Either follow the instruction here
		or
		(if on Debian) check FactoriLabs install script
			Kali Linux
			PiSDR
				E.g. VirtualBox
				`sudo apt install virtualbox-ext-pack`
				With the following Image

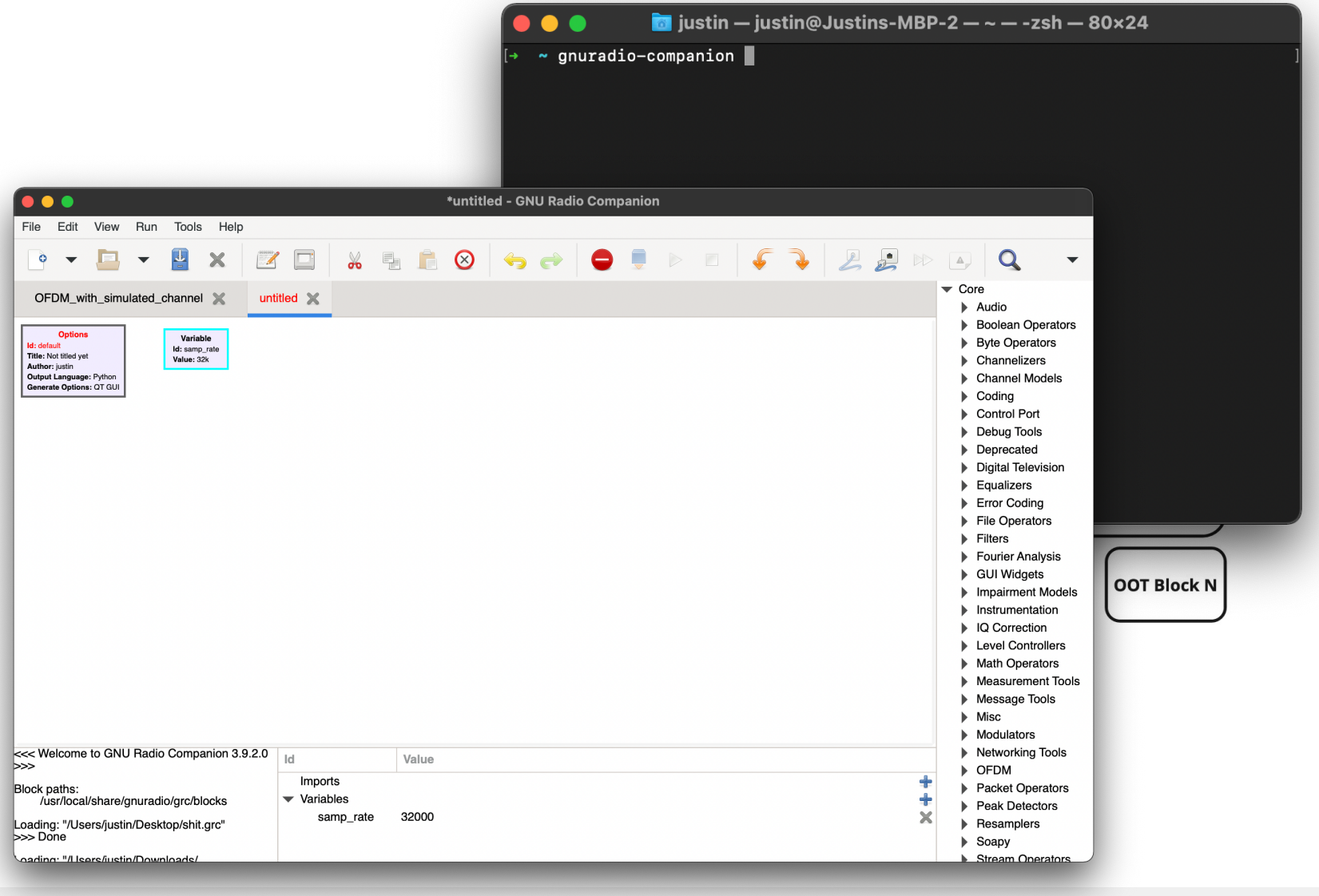
FunKI

*a live system can also be setup given this tutorial

** Apple Silicon builds are nearly impossible right now, with some success reported her

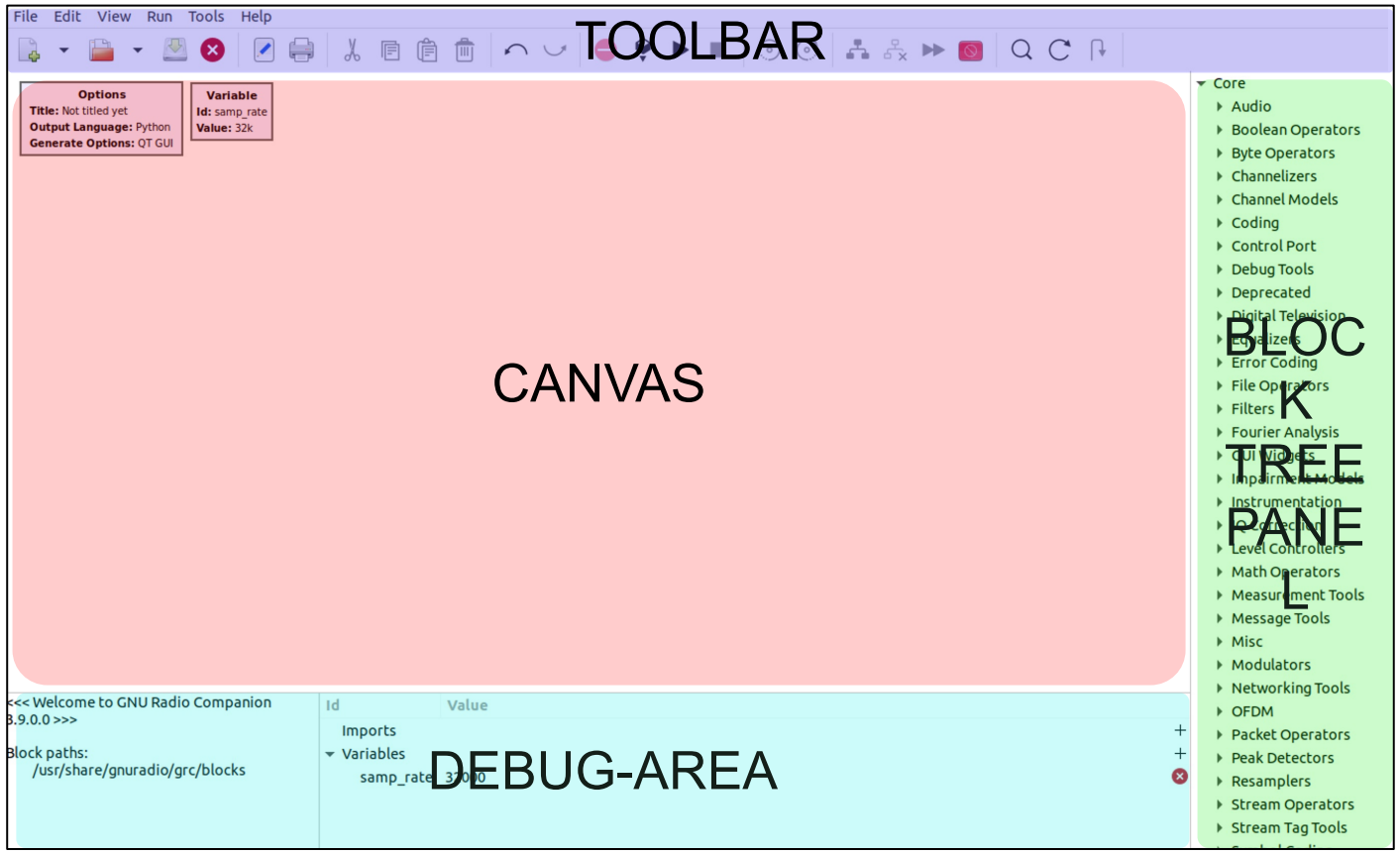
2: Basic concepts

GNUradio 101: GNUradio-companion



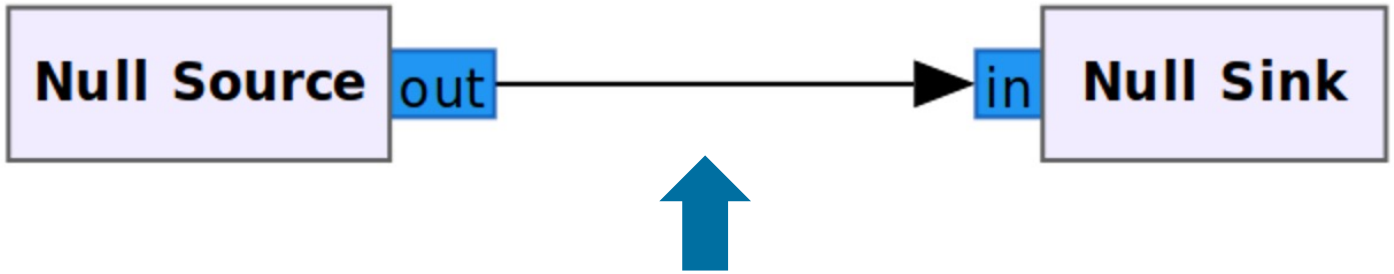
GNUradio companion: The GUI

- Three main areas: TOOLBAR CANVAS, BLOCK TREE PANEL and the “DEBUG-AREA”



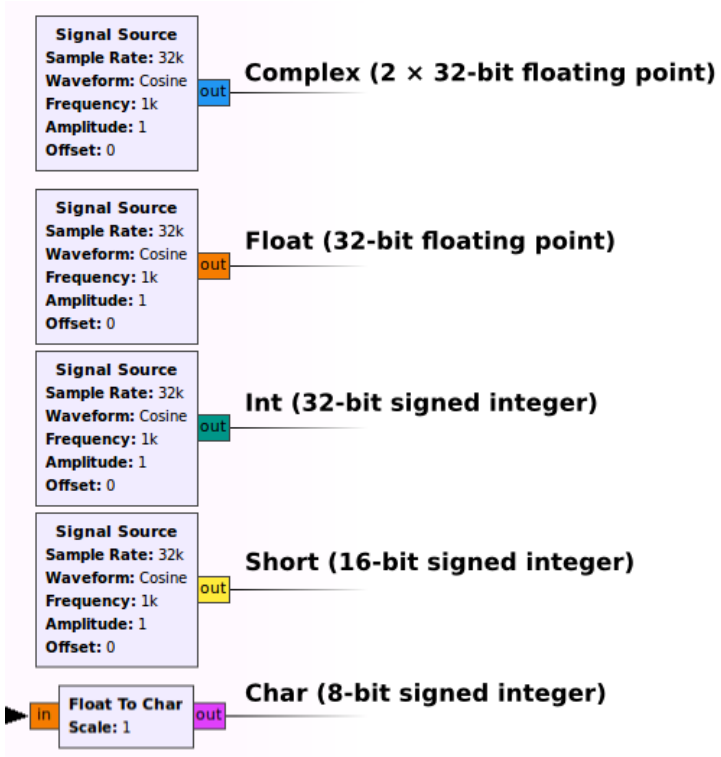
Using GRC: Flowgraph basics

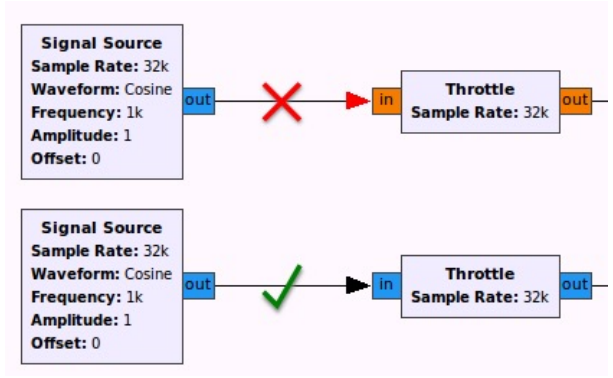
- Basic concept #1: Data flows from source to sink

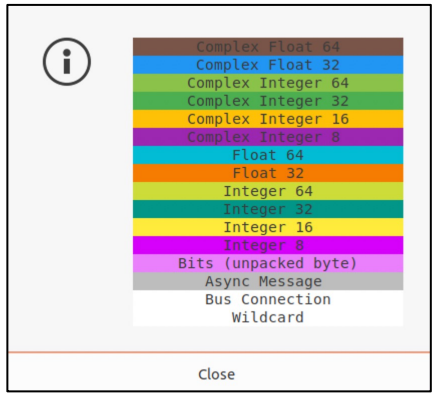


Using GRC: Running the graph

- Basic concept #2: There are several data types, that are not directly compatible with each other







This information can be displayed via the *Help*-section in GRC by clicking on Types

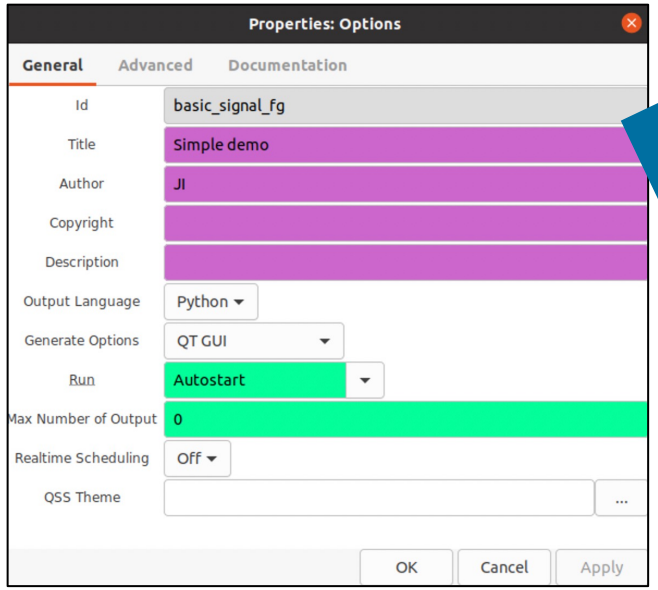
You can change the data type from within the block-settings or by using the arrow-keys

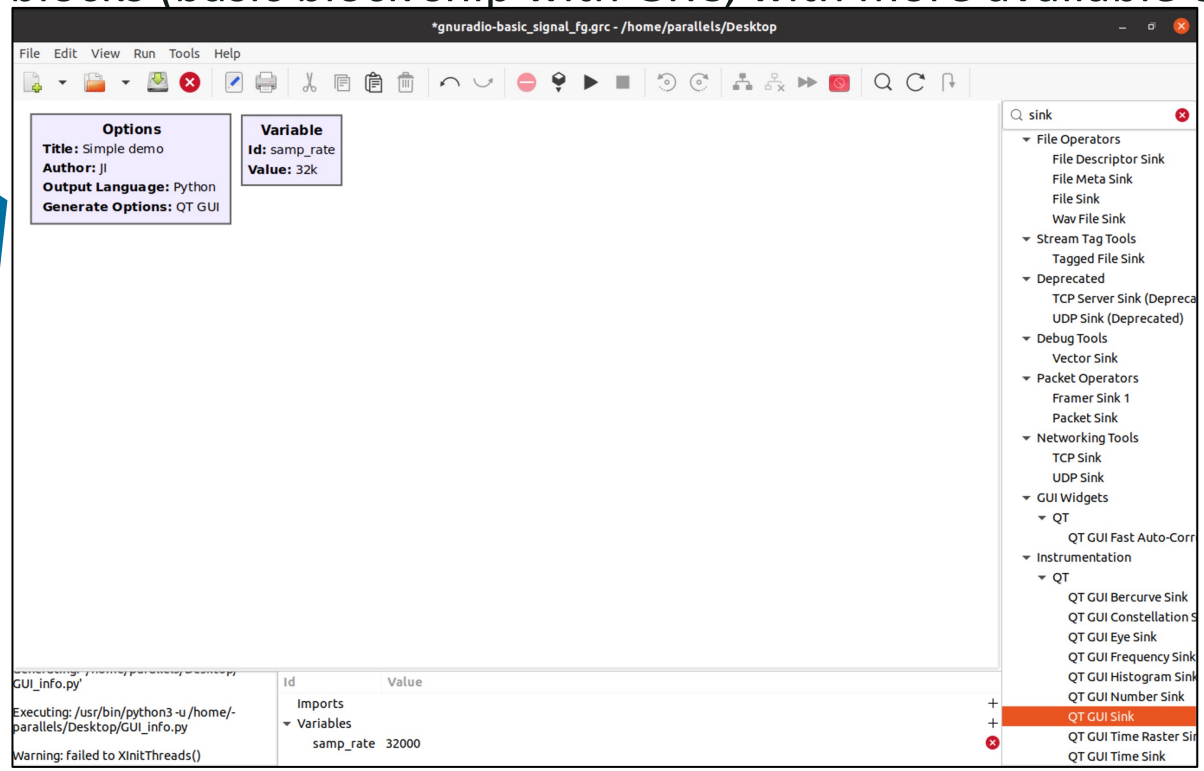
3: Our first flow graph

Using GRC: Initial settings and Graph types

- The first step when creating our first flowgraph is to assign an *ID*/*Title* and check for *Output Language* and *Generation Options* by double clicking on the *Options*-block
  - We will go into further detail on our options here in later slides, for now the settings depicted below are enough

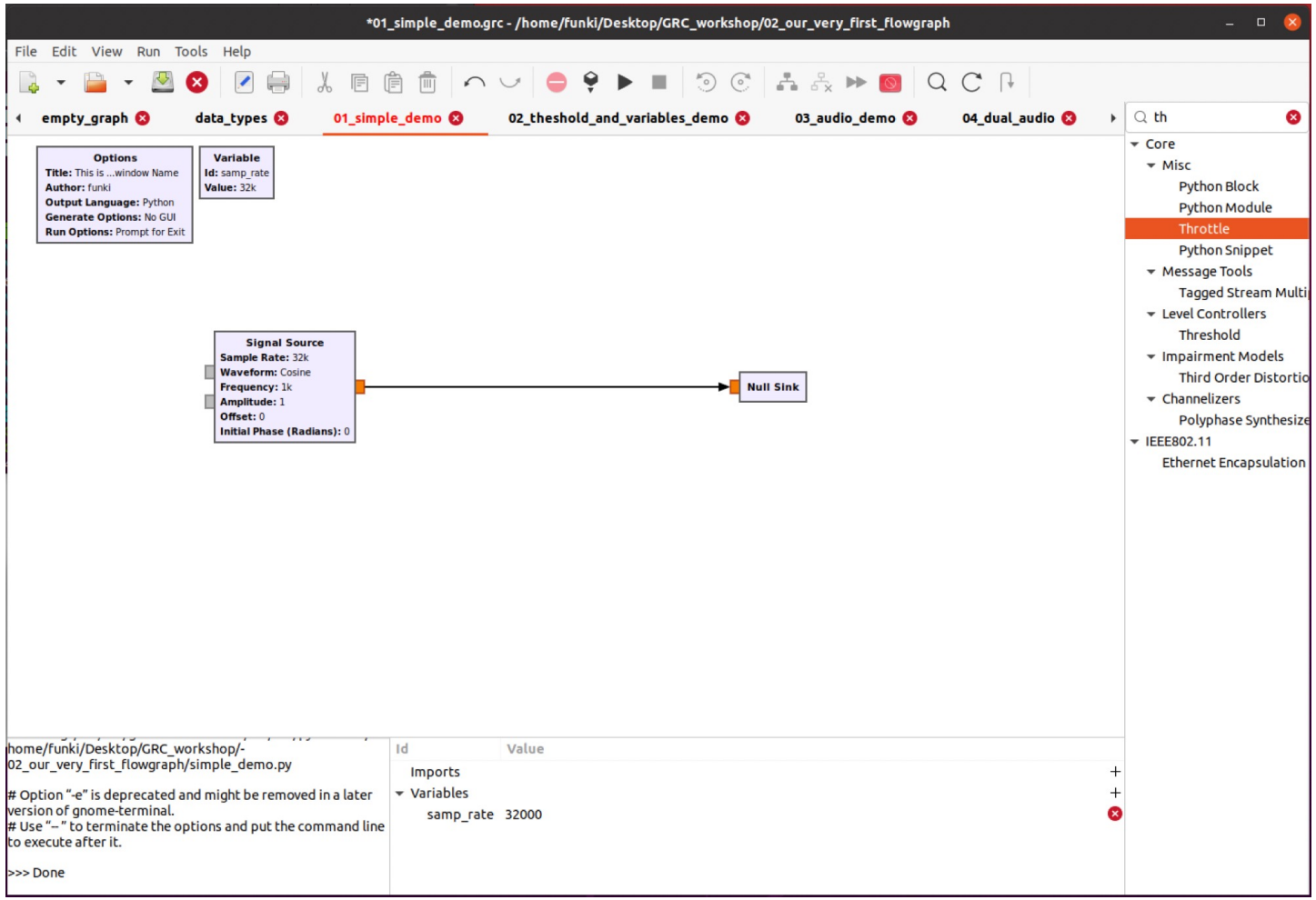
- Flow graphs are composed of different blocks (basic block ship with GRC, with more available online as OOT-modules)





Using GRC: Our very first flowgraph

Basic flowgraph from singal to source



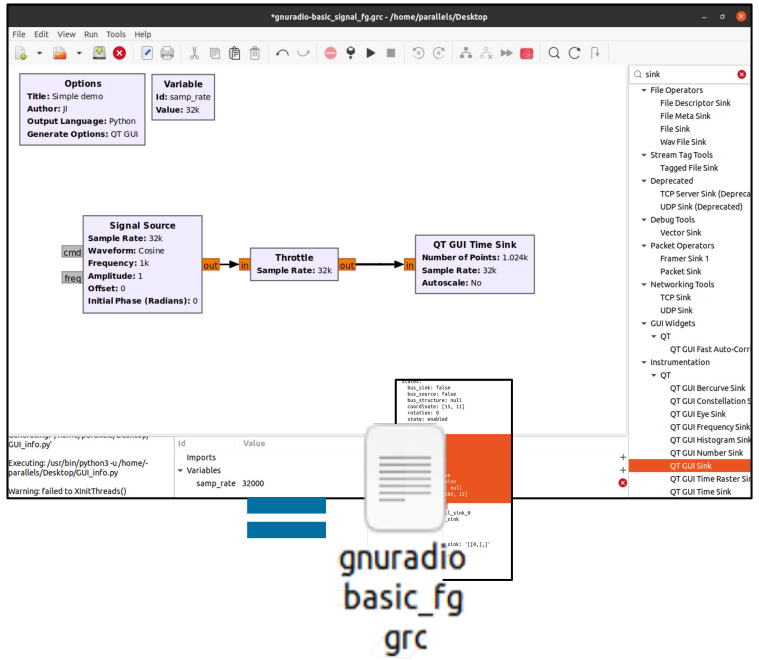
Keyboard Shortcuts

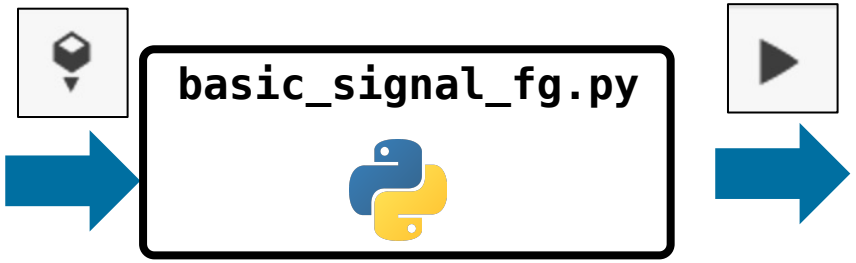
Ctrl+N: Create a new flowgraph.
Ctrl+O: Open an existing flowgraph.
Ctrl+S: Save the current flowgraph or save as for new.
Ctrl+W: Close the current flowgraph.
Ctrl+Z: Undo a change to the flowgraph.
Ctrl+Y: Redo a change to the flowgraph.
Ctrl+A: Selects all blocks and connections.
Ctrl+P: Screen Capture of the Flowgraph.
Ctrl+E: Show variable editor.
Ctrl+F: Search for a block by name.
Ctrl+Q: Quit.
F1 : Help menu.
F5 : Generate the Flowgraph.
F6 : Execute the Flowgraph.
F7 : Kill the Flowgraph.
Ctrl+Shift+S: Save as the current flowgraph.
Ctrl+Shift+D: Create a duplicate of current flow graph.

Ctrl+X/C/V: Edit-cut/copy/paste.
Ctrl+D/B/R: Toggle visibility of disabled blocks or connections/block tree widget/console.
Shift+T/M/B/L/C/R: Vertical Align Top/Middle/Bottom and Horizontal Align Left/Center/Right respectively of the selected block.

Arrowkey Left-Right	Rotate Block

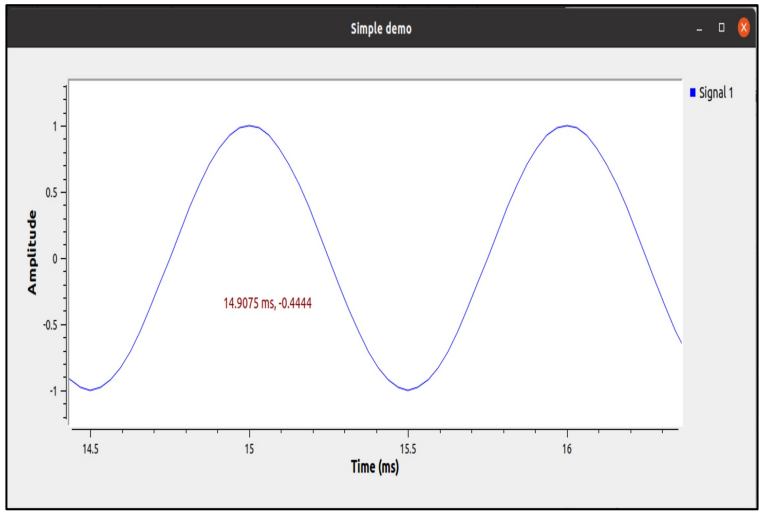
Using GRC: Building and running a graph





"Generate"

"Run"



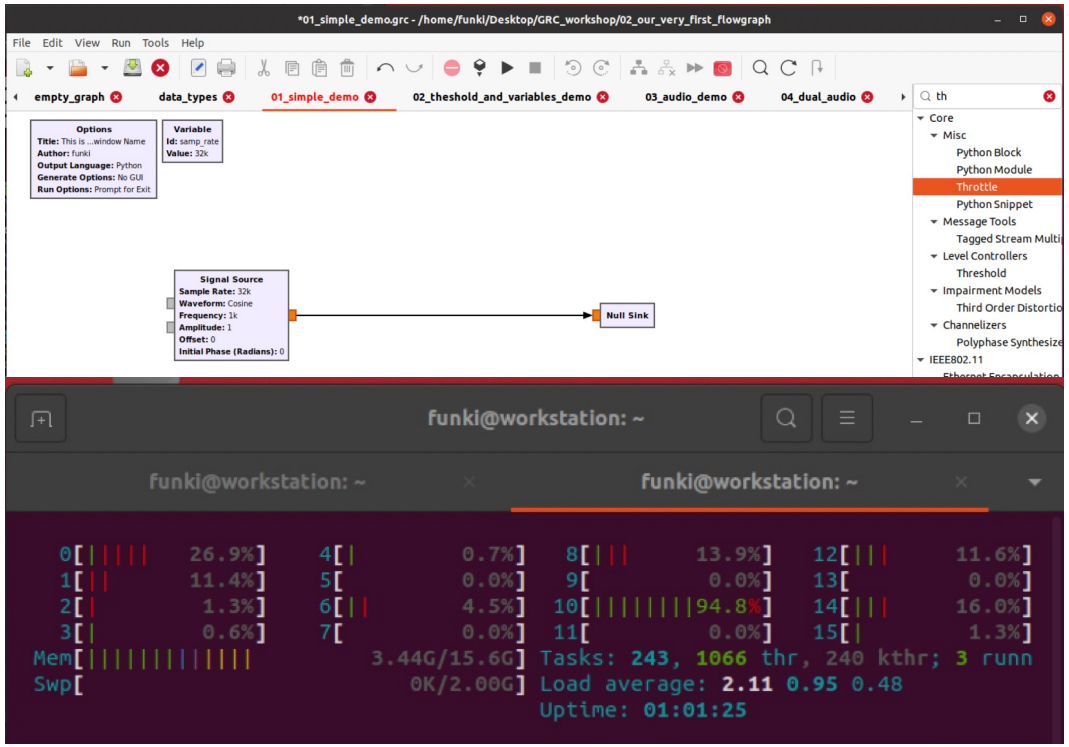
"Terminate"

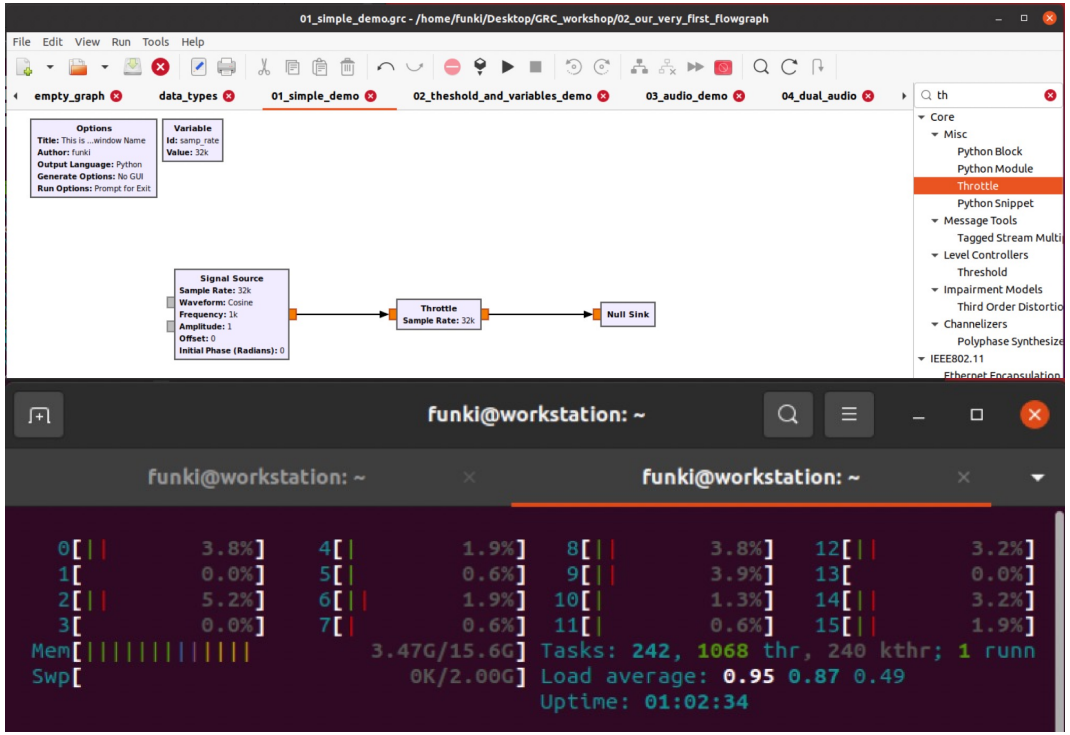
A three-step process

- Creation of a flow-graph in Gnuradio-Companion
- (Python) Code generation
- Running the graph

Using GRC: Running graphs from Terminal -> adding parameters

- Throttle-blocks and why we actually need them

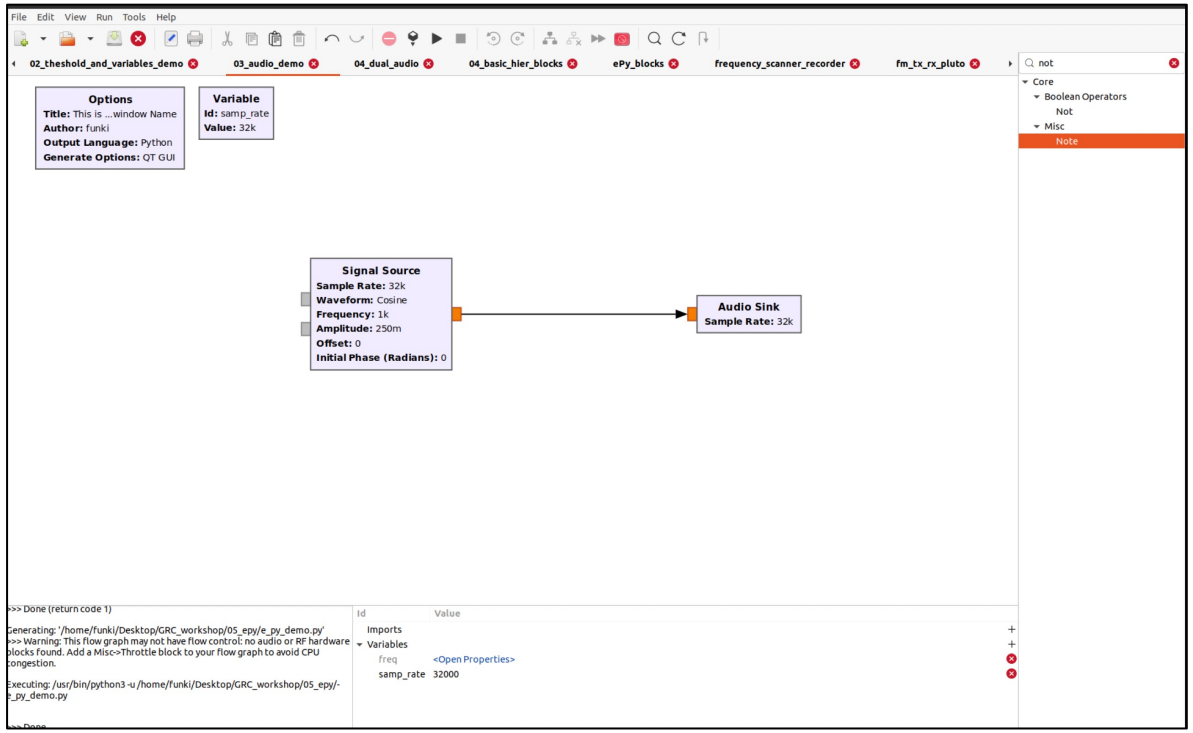




- Graphs via Terminal
- Interacting with them from Terminal-> `python3 your_graph.py -[short_id] --[id] [value]`
  - Refer to this wiki-page for more details

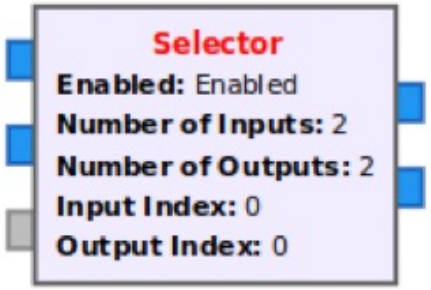
Using GRC: Interacting with Audio

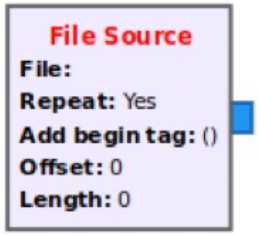
- Finding the necessary Blocks
- Putting them together



Using GRC: File-sources, Selectors and additions

- Variables (QT-sliders)
- Selectors / Combining signals
- Adding noise
- Using file sources





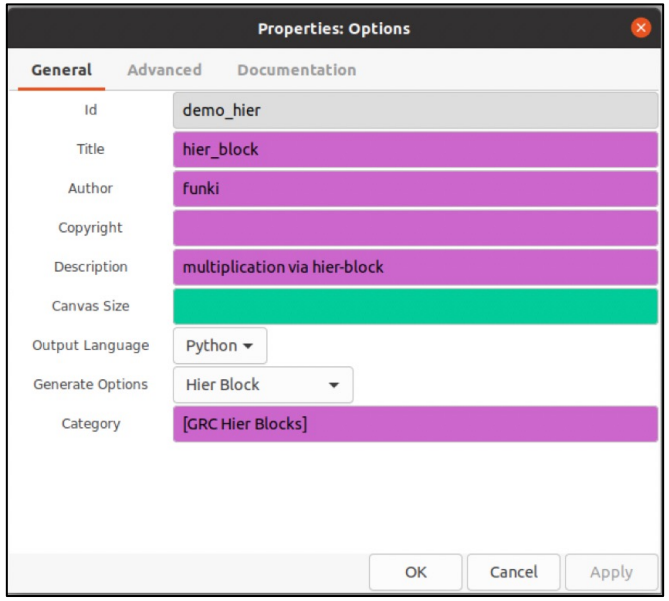
Files can be easily opened using Numpy*:

*instruction in the official FAQ

Hierarchical blocks

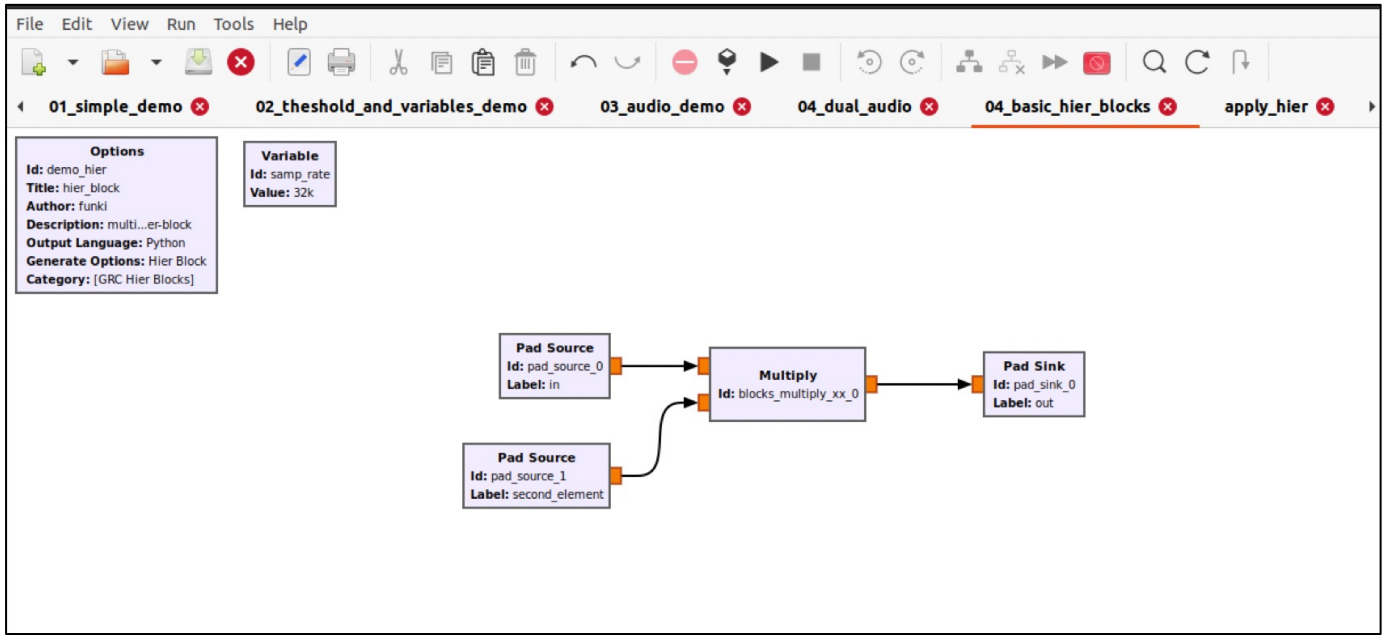
Hierarchical Blocks: Creating a hierarchical block

- Hierarchical blocks are compositions of graph interactions in a abstract ”meta-block”
- They make processing easier and flowgraphs much cleaner
- Enable simple reuse of available subroutines



Main setting

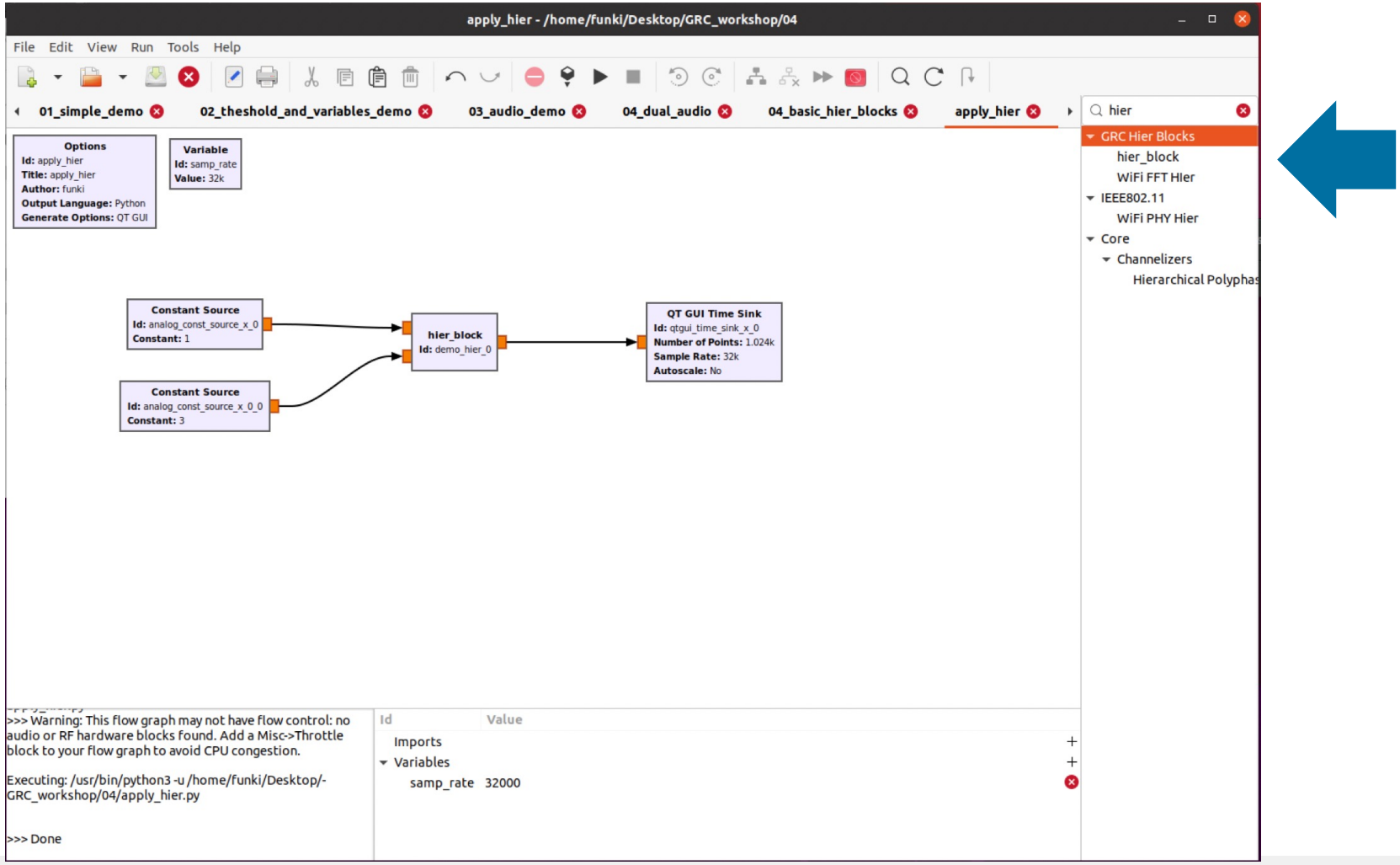
- ID/ Title
- Type
- Generation Options



- Pads (types and other parameters)

Hierarchical Blocks: Usage

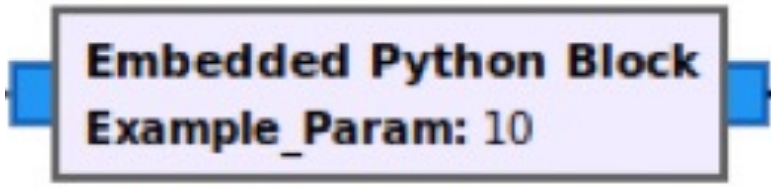
Basically used as any other block



Embedded Python Blocks

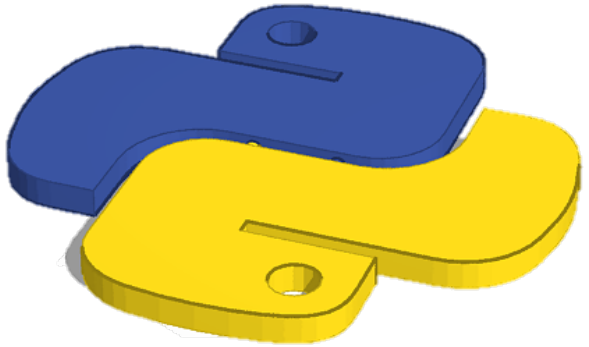
Embedded Python Blocks: The Concept

- Embedded Python blocks (ePy-blocks) are the easiest way to implement custom functionality that goes beyond the available processing possibilities



- Basically, python code within the graph

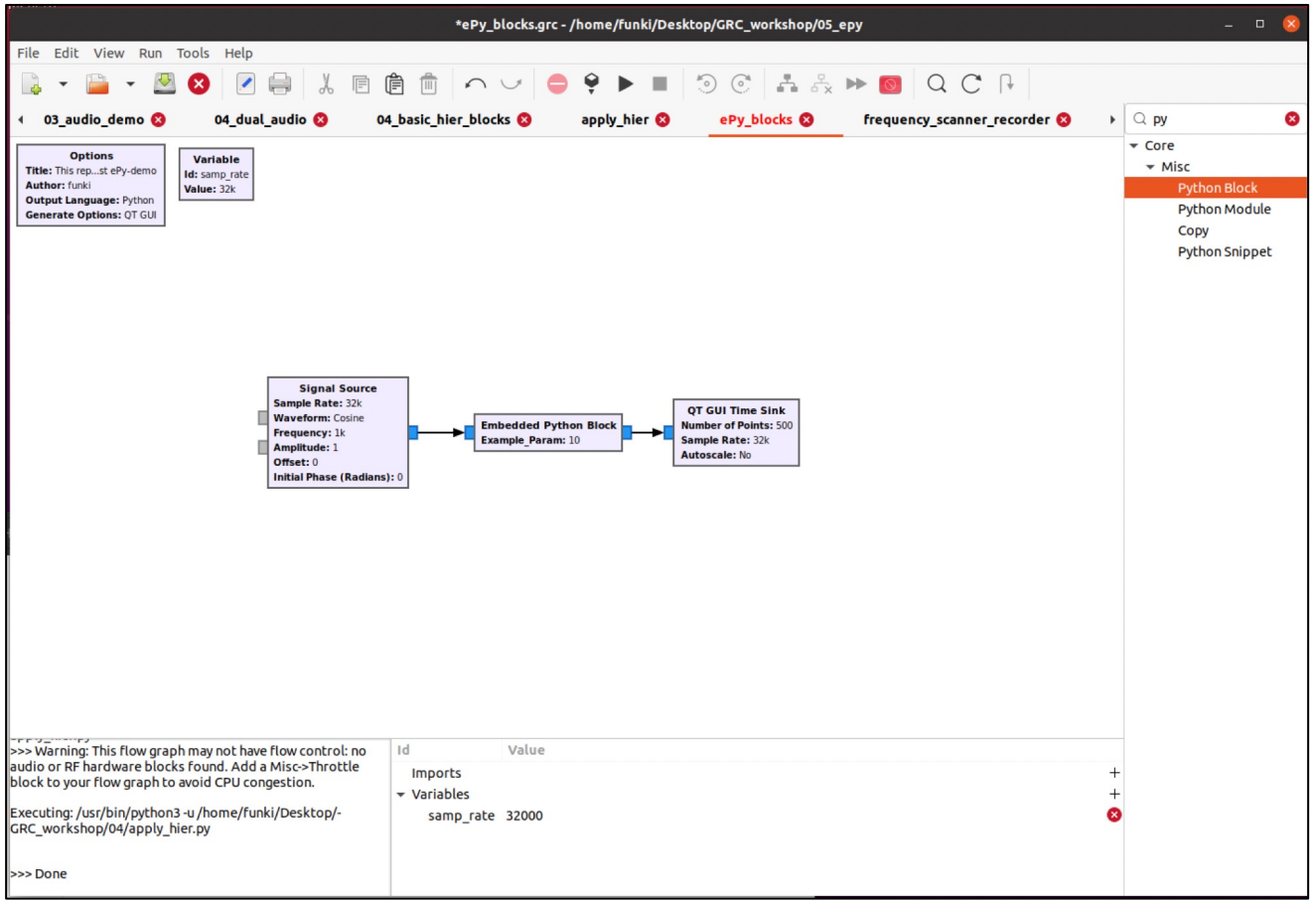
- Inclusion of GNUradio internal as well as third-party-libraries possible



- Fantastic for creation of GUIs and the like

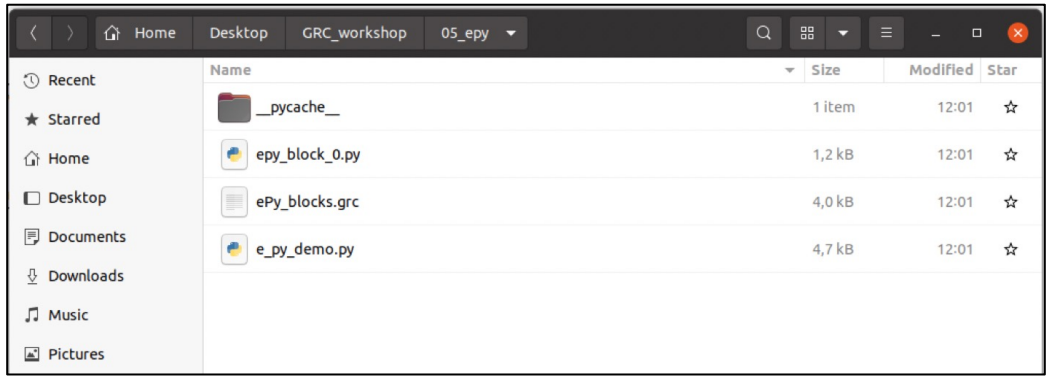
Embedded Python Blocks: Implementation

- Adding an ePy-block
- First look at the code
- Making changes to the code



Embedded Python Blocks: Existing limits

- Downside to the quick implementation is a rather dirty project setup



- Code in theory easy to share but hardly reusable
  -> (need for reimplementation across several flowgraphs)

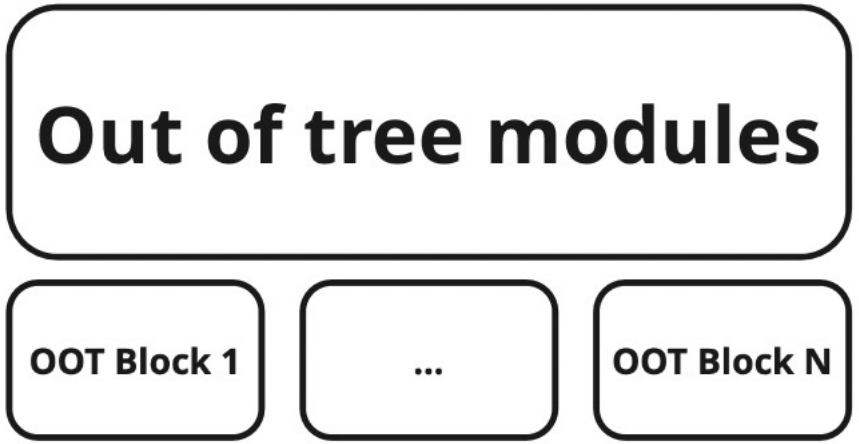
- Speed constraints as python is the only available language…

Embeded Python Blocks are great for prototyping but should not be your first choice!

Sneak preview: Block creation with the grmodtool

GR-Modtool

- Other than using embedded Python Blocks, custom OutOfTree modules are a much cleaner setup
- User can choose between two different programming languages with C++ offering much higher performance
- The full build process is semi-automated and resulting modules can easily be shared
- Also, OOT-module can hold more than a single block
- Detailed step-by-step tutorial found at: https://wiki.gnuradio.org/index.php/OutOfTreeModules



- Block Code in C++/Python
- Build instructions
- C++ Python wrapper setup
- GUI block description
- Documentation
- Examples

GR-Modtool

Creating a module

```
frederikzoels@DESKTOP-OK54VKE:~/cpp_ber_block$ gr_modtool newmod measurement_tools
Creating out-of-tree module in ./gr-measurement_tools...
Done.
Use 'gr_modtool add' to add a new block to this currently empty module.
```

Module Structure

```
frederikzoels@DESKTOP-OK54VKE:~/cpp_ber_block/gr-measurement_tools$ ls
CMakeLists.txt  apps   docs      grc      lib     swig
MANIFEST.md     cmake  examples  include  python
```

Create Block

```
GNU Radio module name identified: measurement_tools
('sink', 'source', 'sync', 'decimator', 'interpolator', 'general', 'tagged_stream', 'hie
r', 'noblock')
Enter block type: general
Language (python/cpp): cpp
Language: C++
Block/code identifier: measure_ber
Please specify the copyright holder: Motius GmbH
Enter valid argument list, including default arguments:
  bool continuous measurement
Add Python QA code? [Y/n] Y
Add C++ QA code? [y/N] N
...
frederikzoels@DESKTOP-OK54VKE:~/cpp_ber_block/gr-measurement_tools$
```

Out of tree module

OOT Block 1

Block types

gr::sync_decimator
gr::sync_block
gr::sync_interpolator
gr::tagged_stream

Language

C++/Python

Copyright

Author..

Block-name

Name in Graph

Arguments

[Optional] List of Args

Additional QA-Code

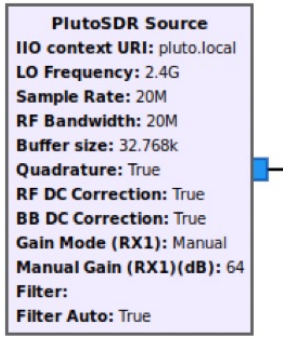
[Optional] Toogle inclusion of QA-code-templates

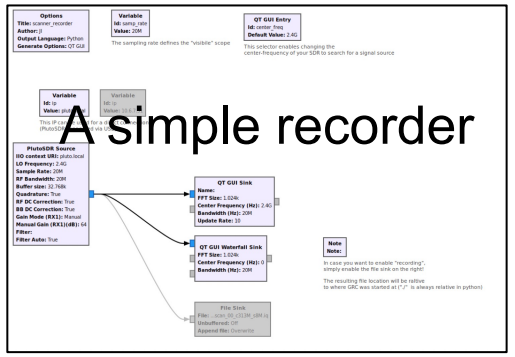
Sneak preview: Interacting with Hardware

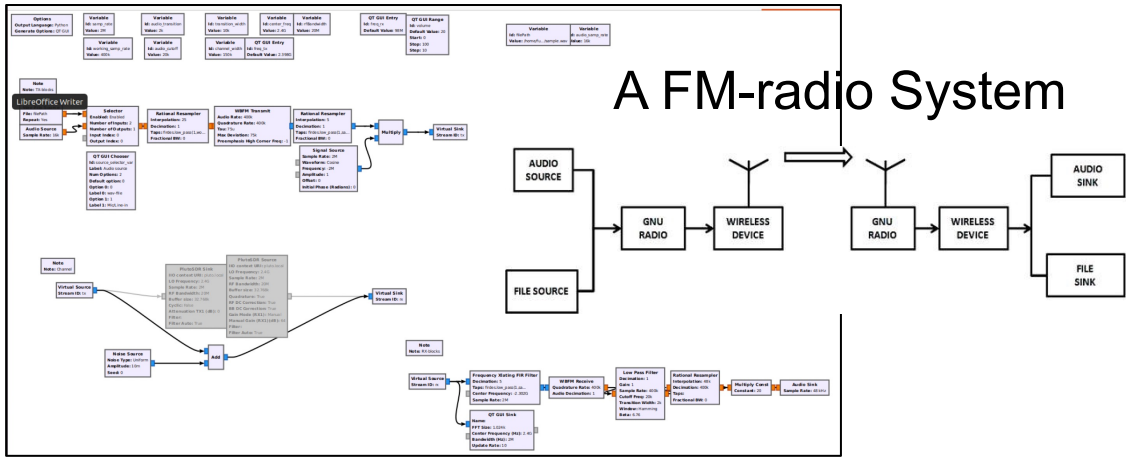
Using GRC: Sneak preview, interfacing with real hardware

- Hardware is addressed the same way any other consume/sink works in GRC
- Different systems are integrated via OOT blocks
- Install Instructions for SDRs are either found on the manufacture's website with some available via tools like pybombs or install scripts
- For the Pluto SDR in use, follow those instructions or make use of the install-script available at FactoriaLabs -> simply run

```
cd install/grc-install/install_scripts ./hackrf_from_source.sh
```







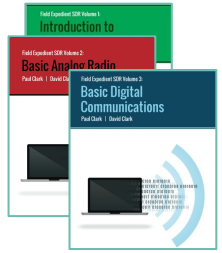
Further materials

Recommended further reading

All of the previous recommendations @

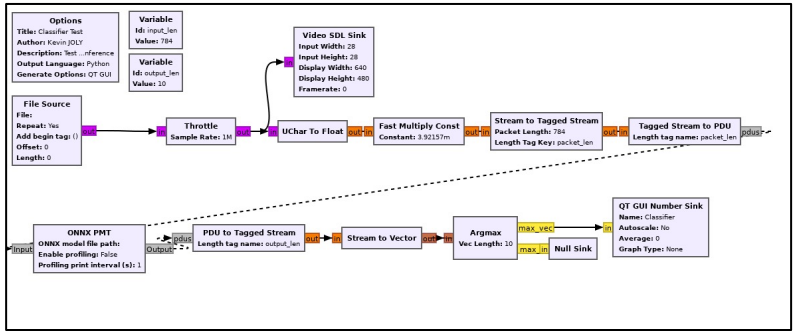
Web presence	Docu	OOTs
Website	Wiki	Library of OOTs
Youtube	Offical Blog	Porting guide
Conference	BastiBl_Blog	
	GRC-list	

Field Expedient SDR Volume 1 to 3 by Paul and David Clark ->Beginners Guide for Hands on GRC and SDRs

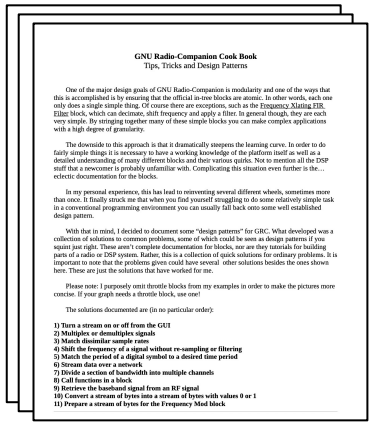


Our Motius GRC-Cookbook

GR-DNN a OOT for Deeplearning with ONNX



Hackaday's amazing cookbook



GNU Radio-Companion Cook Book

Tips, Tricks and Design Patterns

One of the major design goals of GNU Radio-Companion is modularity and one of the ways that this is accomplished is by ensuring that the official in-tree blocks are atomic. In other words, each one only does a single simple thing. Of course there are exceptions, such as the Frequency Xlating FIR Filter block, which can decimate, shift frequency and apply a filter. In general though, they are each very simple. By stringing together many of these simple blocks you can make complex applications with a high degree of granularity.

The downside to this approach is that it dramatically steepens the learning curve. In order to do fairly simple things it is necessary to have a working knowledge of the platform itself as well as a detailed understanding of many different blocks and their various quirks. Not to mention all the DSP stuff that a newcomer is probably unfamiliar with. Complicating this situation even further is the… eclectic documentation for the blocks.

In my personal experience, this has lead to reinventing several different wheels, sometimes more than once. It finally struck me that when you find yourself struggling to do some relatively simple task in a conventional programming environment you can usually fall back onto some well established design pattern.

With that in mind, I decided to document some "design patterns" for GRC. What developed was a collection of solutions to common problems, some of which could be seen as design patterns if you squint just right. These aren't complete documentation for blocks, nor are they tutorials for building parts of a radio or DSP system. Rather, this is a collection of quick solutions for ordinary problems. It is important to note that the problems given could have several other solutions besides the ones shown here. These are just the solutions that have worked for me.

Please note: I purposely omit throttle blocks from my examples in order to make the pictures more concise. If your graph needs a throttle block, use one!

The solutions documented are (in no particular order):

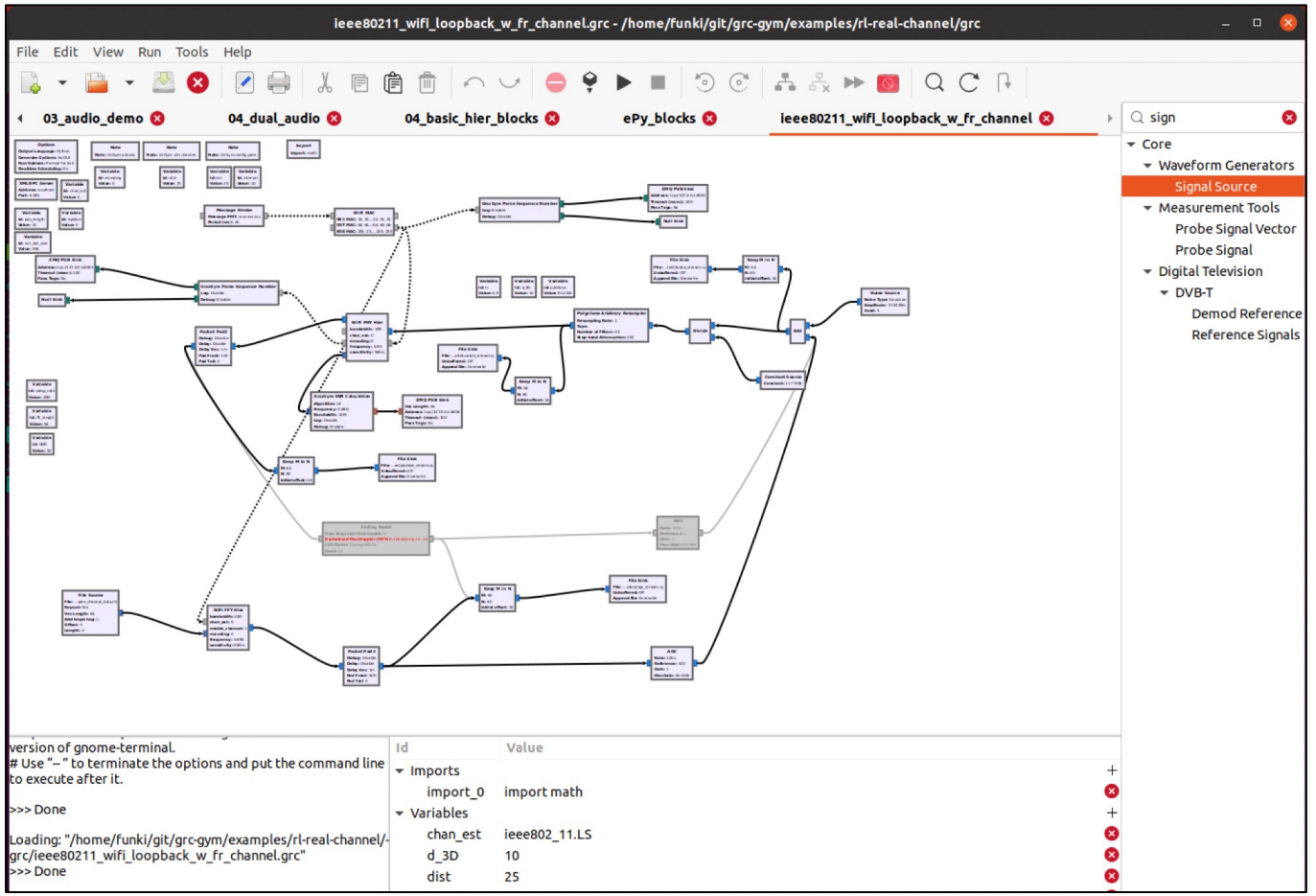
1) Turn a stream on or off from the GUI
2) Multiplex or demultiplex signals
3) Match dissimilar sample rates
4) Shift the frequency of a signal without re-sampling or filtering
5) Match the period of a digital symbol to a desired time period
6) Stream data over a network
7) Divide a section of bandwidth into multiple channels
8) Call functions in a block
9) Retrieve the baseband signal from an RF signal
10) Convert a stream of bytes into a stream of bytes with values 0 or 1
11) Prepare a stream of bytes for the Frequency Mod block

Q&A

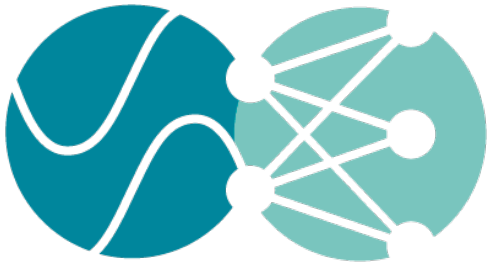
Next time more details on RL in GRC

Presentation of Florian Geiser's MA-thesis on 30.9.2021

- Implmented in GRC 3.8
- Based on a OOT module for reinforcement learning
- Makes use of the Datarecodring provided by the consortium!



See: https://gitlab.com/librespacefoundation/sdrmakerspace/gr-dnn and the related paper "Deep learning inference in GNU Radio with ONNX" By Rodriguez et. Al.



FunKI


CREONIC
ip cores & system solutions


Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH


intel


MOTIUS
WE R&D.


NOKIA


TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN


Universität
Bremen


University of Stuttgart
Germany

- **Task 1.c.1:** Anwedungsfälle und Szenarien
- **Task 2.a.1**: Analyse von verschiedenen ML-Ansätzen für den Einsatz bei der Signaldetektion (zur Laufzeit)
- **Task 2.a.2**: MIMO Kanalschätzung und -entzerrung durch neuronale Netze
- **Task 2.a.3**: *Adaption von MIMO-OFDM Systemen mittels neuronaler Netze*
- **Task 2.c.1**: *Untersuchung von geeigneten ML-Verfahren für die Signalverarbeitungskette von Mehrträgermodulationsverfahren*
- **Task 2.c.2**: *Training der Neuronalen Netze und Simulation des Gesamtsystems*
- **Task 2.c.3**: *Entwurf und Implementierung des End-to-End Systems durch Auto-Encoder*
- **Task 3.a.1**: Analyse von geeigneten Verfahren für die KI-basierte Kanaladaption und Entwicklung einer Simulationsumgebung
- **Task 3.a.2**: Implementierung der Kanaladaption basierend auf den geschätzten und vorhergesagten Kanalparametern
- **Task 3.b.1**: Evaluierung geeigneter ML-Verfahren für die gemeinsame Kanalparameterschätzung
- **Task 3.b.2**: Entwicklung des KI-Verfahrens für die gemeinsame Parameterschätzung
- **Task 3.d.1**: Radio Traffic Sequenz Detektion und Interferenz Management für dynamische Kanalzugriffsverfahren
- **Task 3.d.2**: Entwurf einer Simulationsumgebung für den Test von dynamischen Kanalzugriffsverfahren