



 **ADI** **FTC**

COLLABORATE • CREATE • ACCELERATE

# Software hands-on training kit: What software do I need for my SDR?

**Mihai Bancisor, Valentin Beleca**

SSG/CSE (Customer Solutions Enablement)

# Agenda

## Introduction

## SDR (Software Defined Radio)

**Ex. 1:** IIO Oscilloscope

**Ex. 2:** Transmit and receive a complex sinusoid with GNU Radio

**Ex. 3:** Transmit and receive a complex sinusoid with Python

**Ex. 4:** How to explore and tweak device attributes

**Ex. 5:** Doppler Radar with GNU Radio

**Ex. 6:** Phase Shift Keying – BPSK in Python

**Ex. 7:** Phase Shift Keying – QPSK in GNU Radio

**Ex. 8:** Phase Shift Keying – Receive a message in GNU Radio

**Ex. 9:** Phase Shift Keying – Receive a message and store it in a file

**Ex. 10:** Amplitude Shift Keying – GNU Radio Example

**Ex. 11:** BPSK without additional digital processing – GNU Radio

**Ex. 12:** QPSK – Constellation Modulator in GNU Radio

**Ex. 13:** QPSK – Frequency Locked Loop in GNU Radio

**Ex. 14:** QPSK – Symbol Sync in GNU Radio – Hands On  
QPSK – Costas Loop in GNU Radio

**Other open-source software platforms**

**Conclusions**

# Introduction

# Software hands-on training kit

**Scope:** a program aimed at providing application specific (e.g. Instrumentation, SDR, Process control & Factory Automation, Connectivity,...) software trainings covering the relevant software technologies for that application.

**Principle:** anyone can get all the necessary hardware from *analog.com*, download the software from *ADI's github*, and access the training material on *ADI's github documentation pages* to do the training anywhere, anytime without the need of custom hardware.

**Available trainings:** <https://analogdevicesinc.github.io/documentation/learning/index.html>



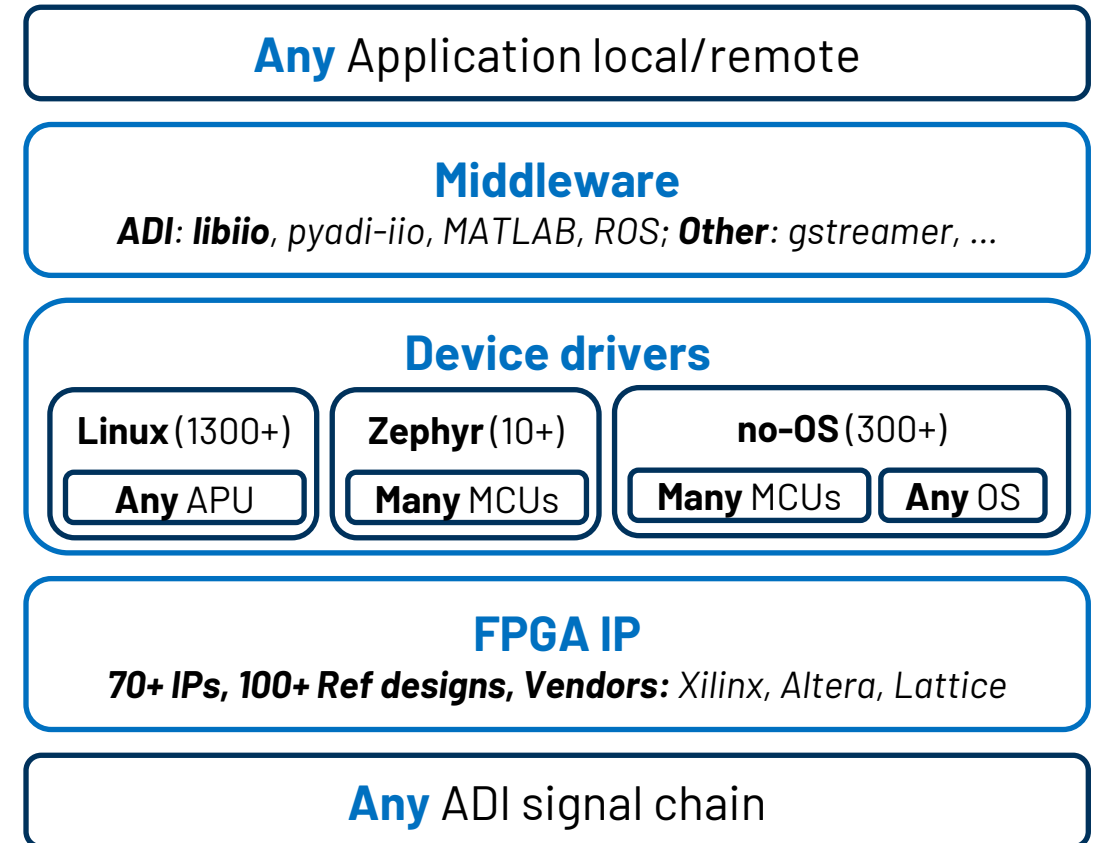
# Any *signal chain* to Any *application* on Any *platform*

Multi-platform device drivers and FPGA IP enable interfacing any signal chain with any MCU/SoC on any OS

One common middleware enables any application to work seamlessly with any signal chain on any compute platform

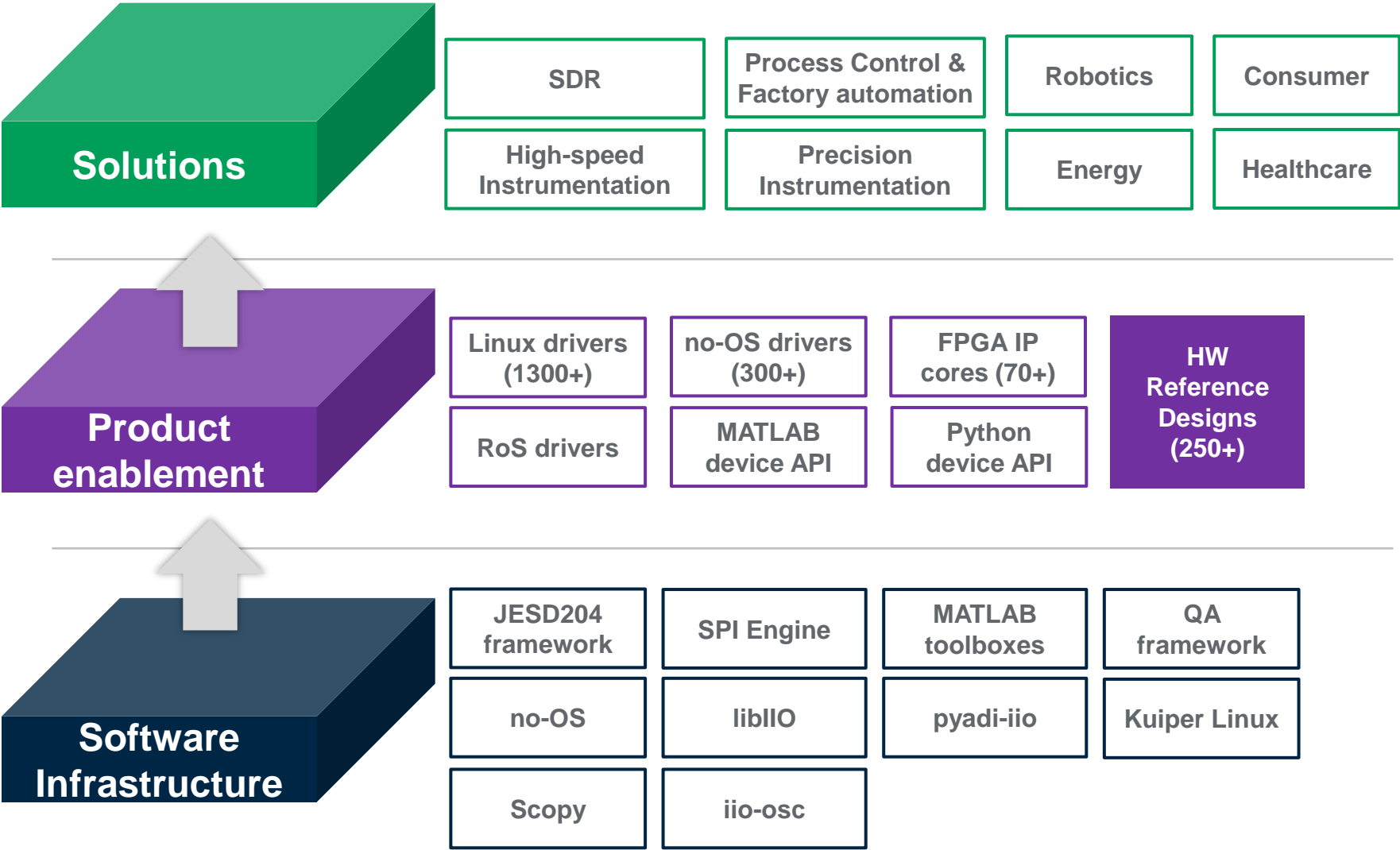
## ADI Software Strategy, Rob Oshana, 2023

- “Our vision is to be a **world-class software provider**: software is increasingly important to ADI and our customers”
- “That begins by building a solid foundation: resilient & secure supply chain, standard level of quality, consistent enablement & tooling, **integrated hardware/software co-design, using open-source communities** and partners”



**ADI has the most comprehensive and scalable offering in the industry**

# SSG/CSE (Customer Solutions Enablement)

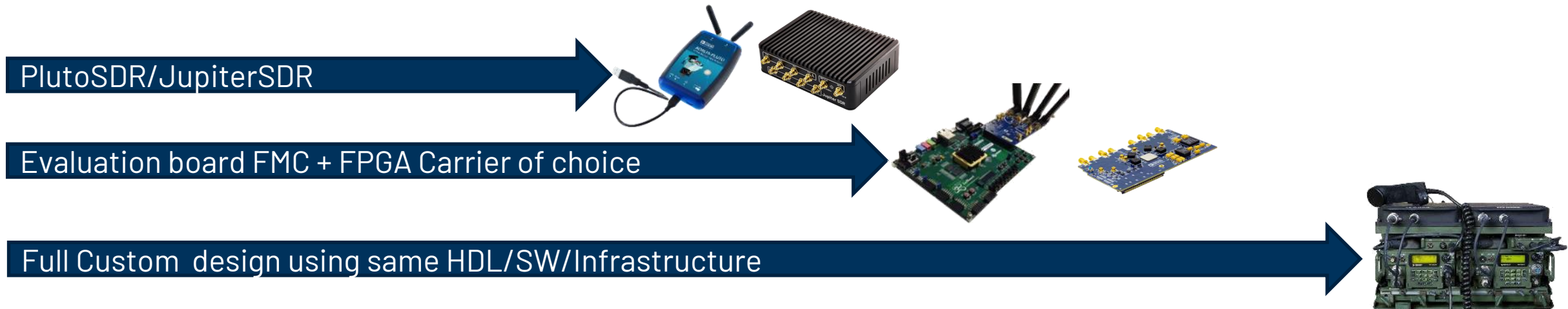
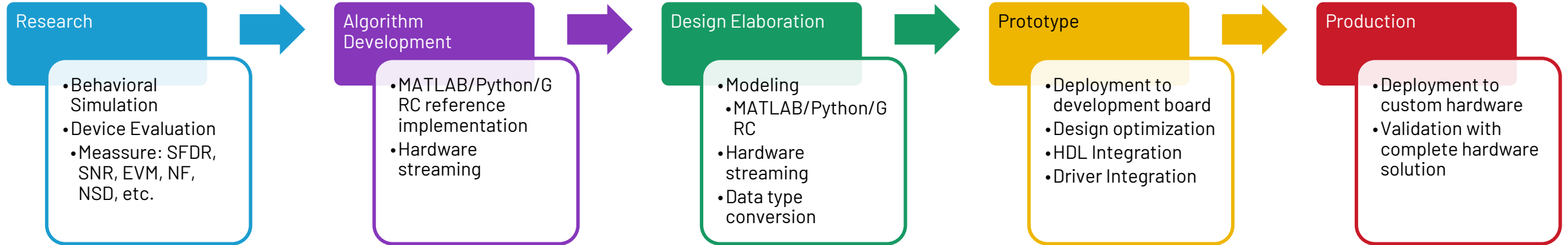


 **GitHub**  
 **ADI EngineerZone**  
SUPPORT COMMUNITY  
 **ANALOG DEVICES** | Wiki  
AHEAD OF WHAT'S POSSIBLE™

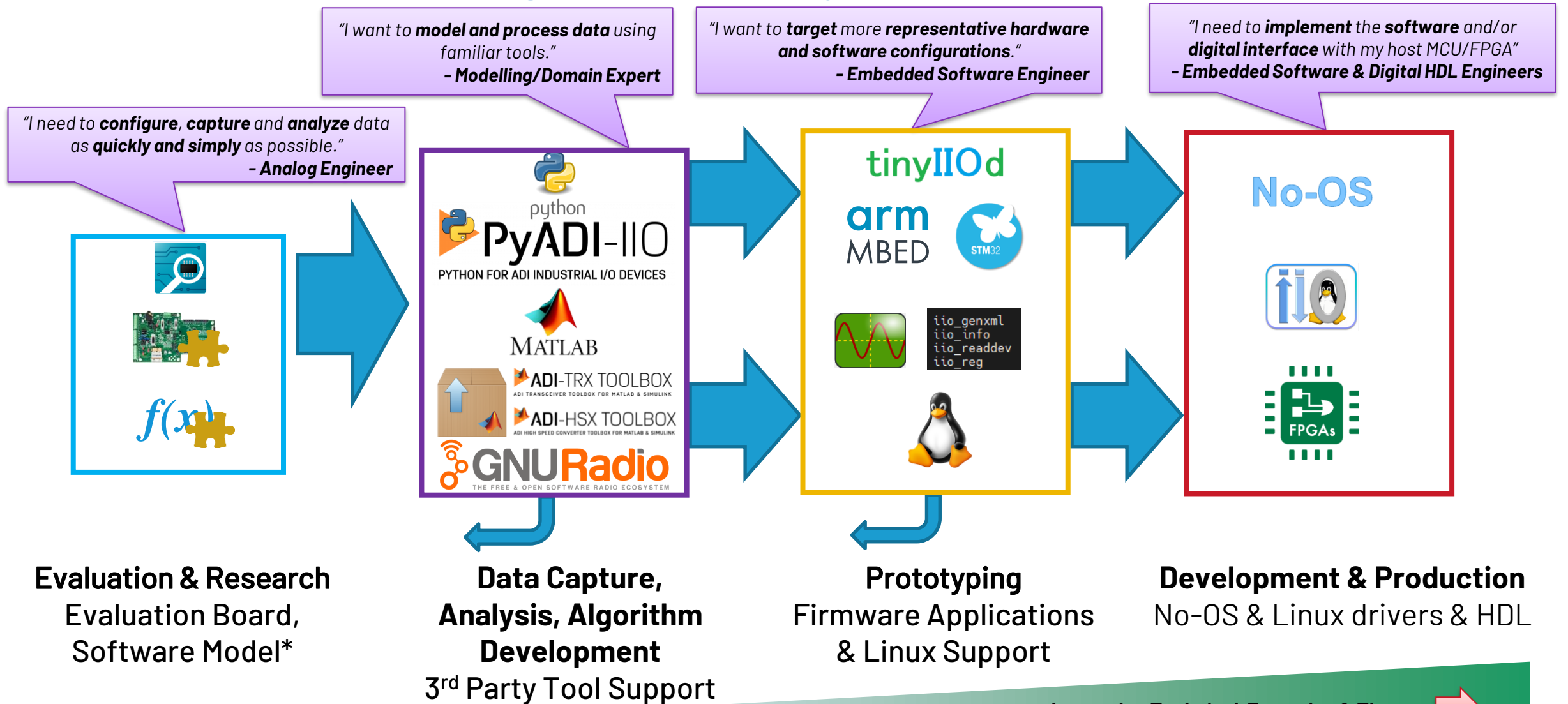
**Customer Support**

ADI Analog & SW FAEs

# Typical Customer Design Flow



# Software in the Design-in Journey

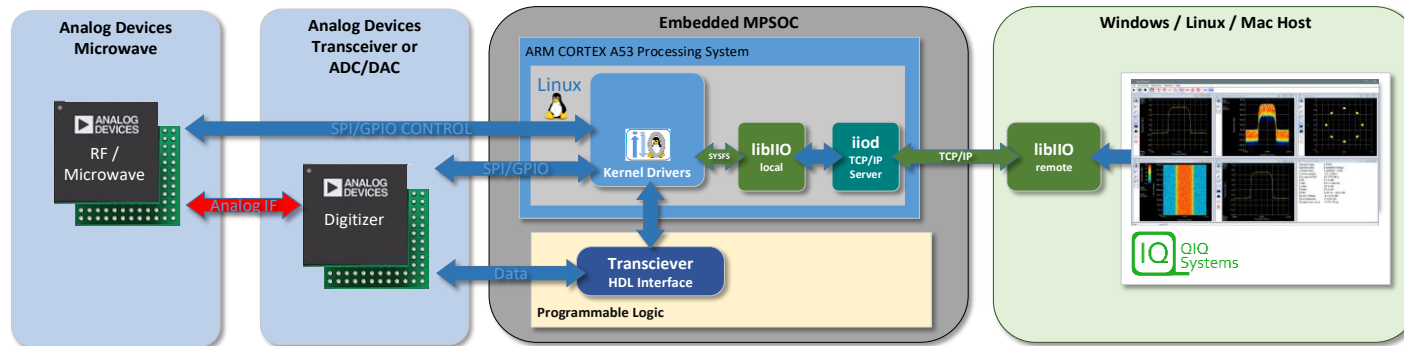
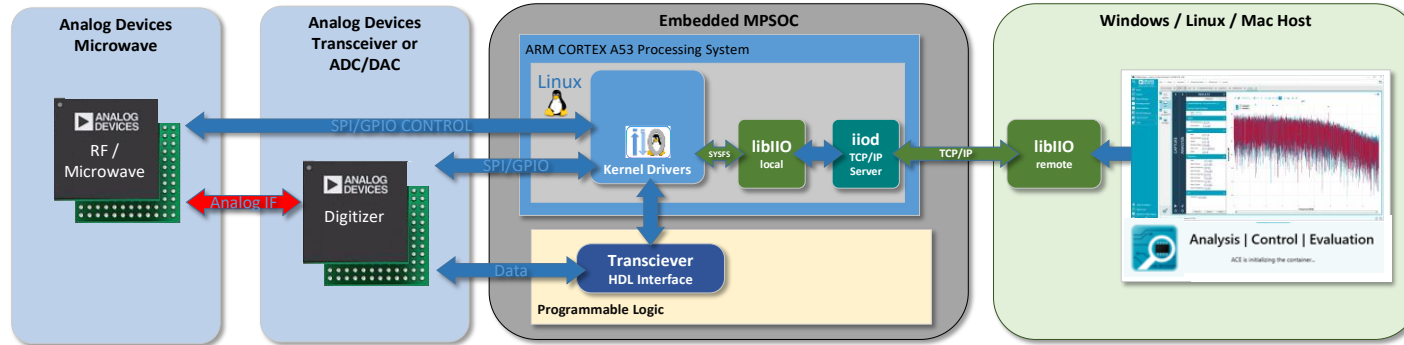
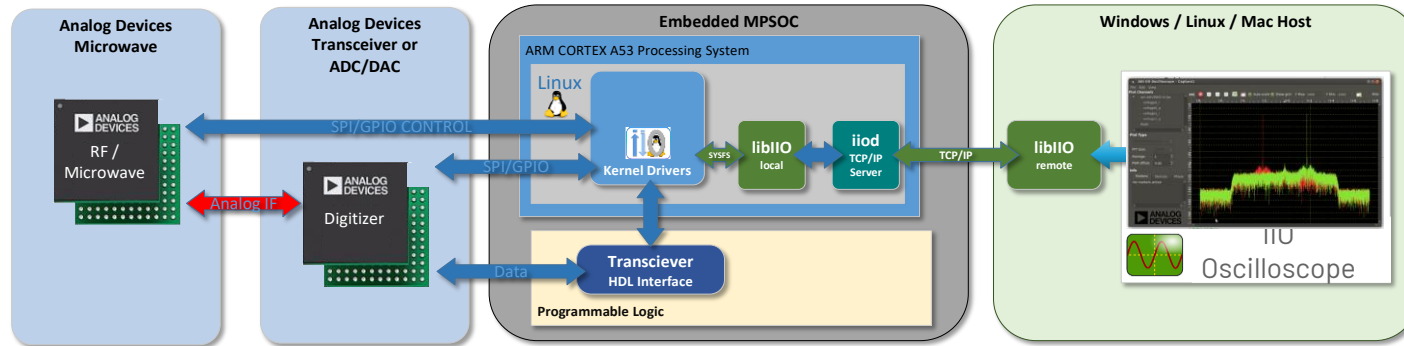


Increasing Technical Expertise & Time



\* Software Model is optional

# Evaluation, Test and Analysis

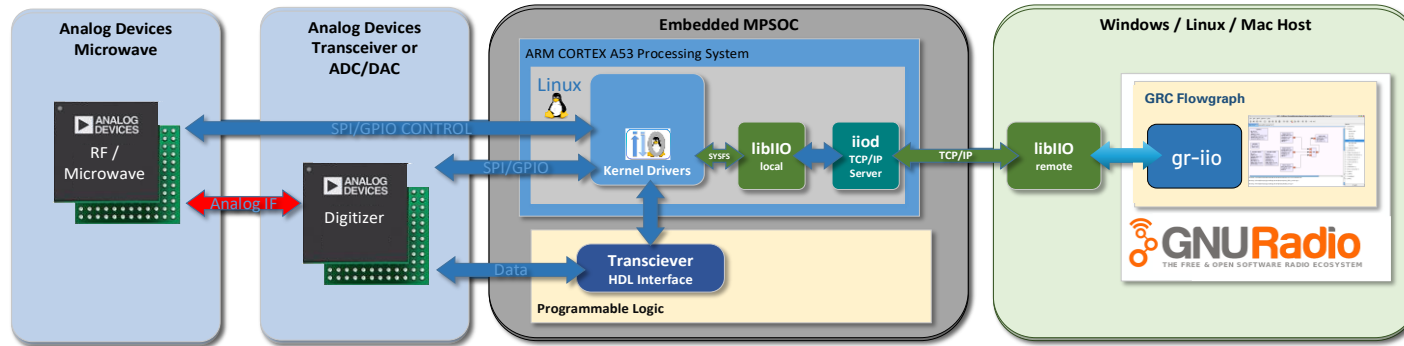


**Single cohesive  
software solution -  
meeting customers  
in their ecosystem  
or at their tools of  
choice**

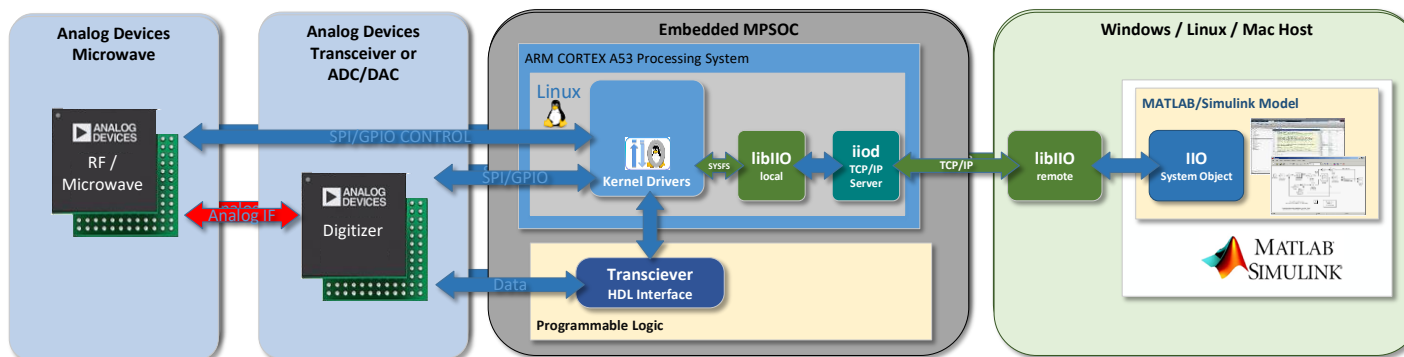
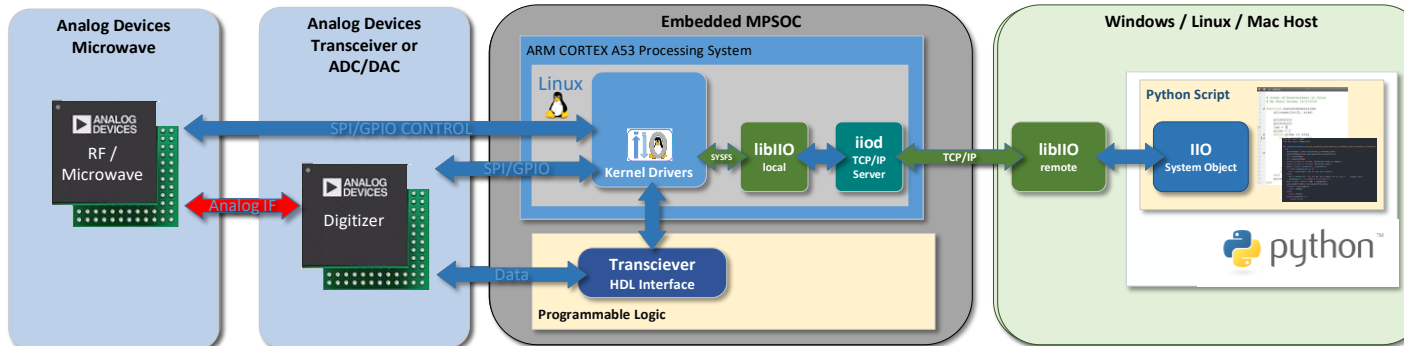
## ❑ Product Evaluation

- ❑ Using Hardware & Software Components to confirm that the Converter meets the Application needs
- ❑ Time is (very roughly) proportional to complexity and how application specific it needs to be

# Algorithmic Development, Modeling, Prototyping



**Single cohesive  
software solution -  
meeting customers  
in their ecosystem  
or at their tools of  
choice**



- ❑ **Product Prototyping**
  - ❑ Plug 'n' Play hardware and software, see the key features/performance of the part
  - ❑ Configure, Capture signals or Generate waveforms in 10-15 minutes



# Building Blocks for development and new revenue streams

Hardware Development

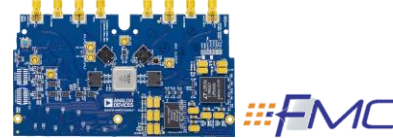
## Evaluation/Prototyping

Open Market Development platforms  
(Off the shelf carrier boards)



FMC Compatible Dev platforms

ADI Evaluation boards  
(Daughter Boards)



Variety of FMC Compatible Boards  
MxFE, Navassa, Talise, Catalina, Madura

Open Source & Bare Metal  
Device level Drivers



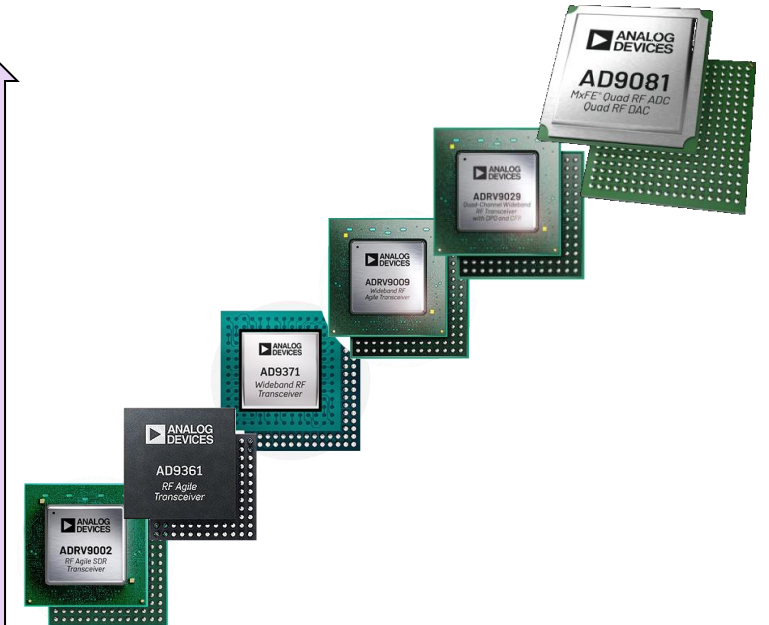
ADI LibIIO

ADI IIO-Scope



## Options for different RF Applications

Channel Bandwidth  
40MHz 70MHz 200MHz 2GHz



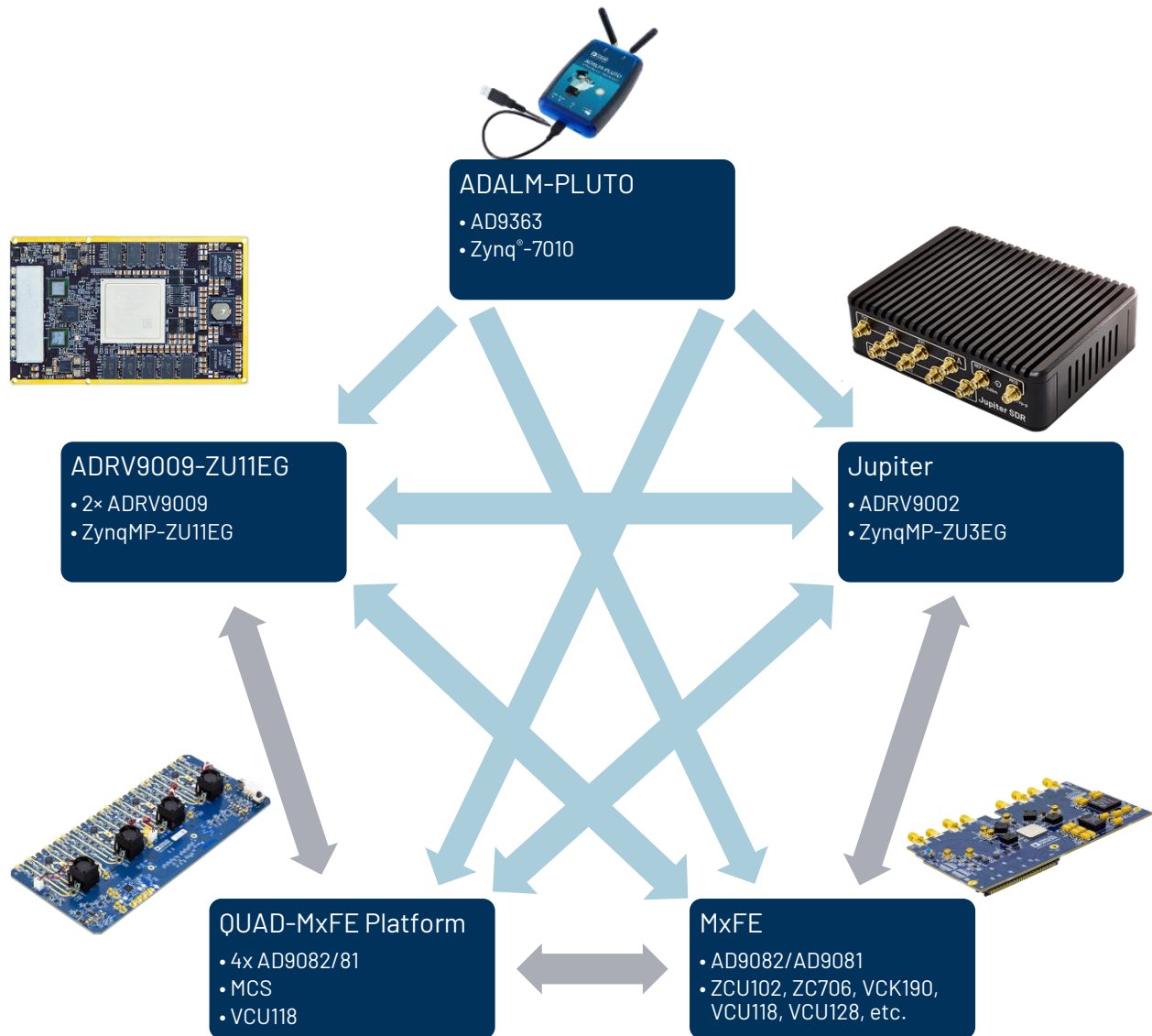
LVDS/CMOS  
61.44MSPS  
2 Channel

JESD204B  
491MSPS  
2 Channel

JESD204B/C  
12GSPS/4GSPS  
4 Channel

Channel Sampling Speed

# Common Architecture Makes It Easy to Transition Between Platforms



- Shares same software/HDL/hardware stack
  - Makes it easy to move from one to the other
  - Differentiated on form factor, number of channels, connectivity, expandability, FPGA resources, CPU resources
- Start with ADALM-PLUTO
  - Stream to MATLAB®, Simulink®, or GNU Radio via USB
  - Take data in the field
  - Validate your communication, radar, or SIGINT algorithms in MATLAB, Simulink, or GNU Radio
  - Start moving to embedded signal processing
    - Transition to production-ready SOM
    - Use custom chip-down design
- Same tools, same libraries, same HDL
  - Vivado, MATLAB, IIO work the same on all platforms
  - Common HDL at [github.com/analogdevicesinc/hdl](https://github.com/analogdevicesinc/hdl)
  - Common Linux® kernel at [github.com/analogdevicesinc/linux](https://github.com/analogdevicesinc/linux)

# What is Kuiper

ADI Kuiper Linux is an open-source Linux Distribution based on Raspberry Pi OS



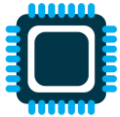
It is the primary distribution for product evaluation boards and reference designs



It **includes pre-built boot files, device drivers** and a variety of **development utilities**



The first release was published at the beginning of 2021



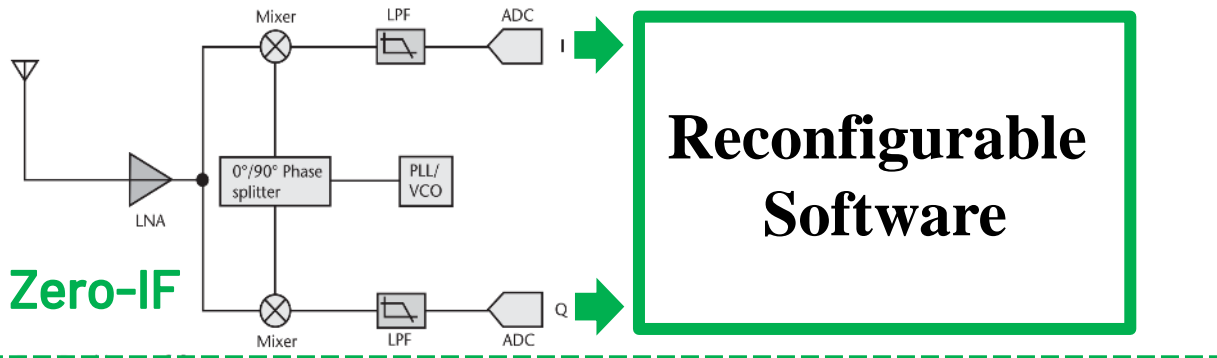
**Supports over 120 FPGA-based projects and over 25 Raspberry Pi-based designs**

# SDR Hardware



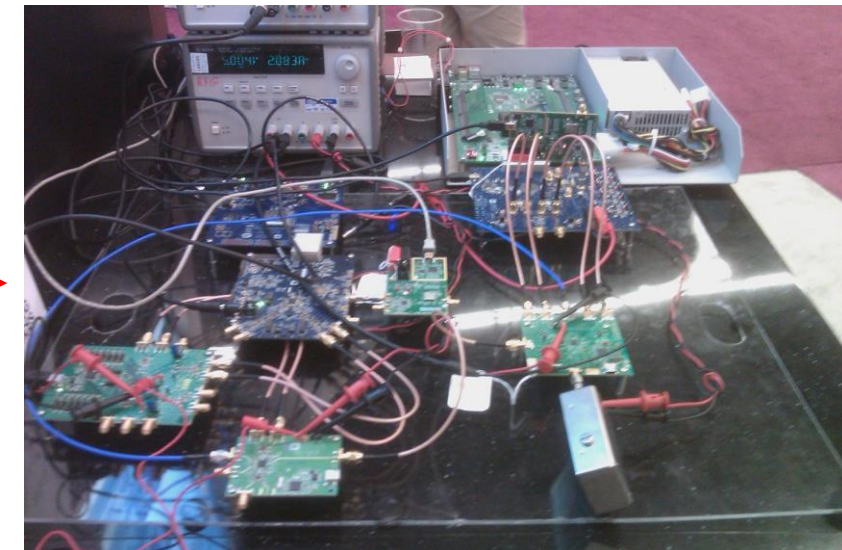
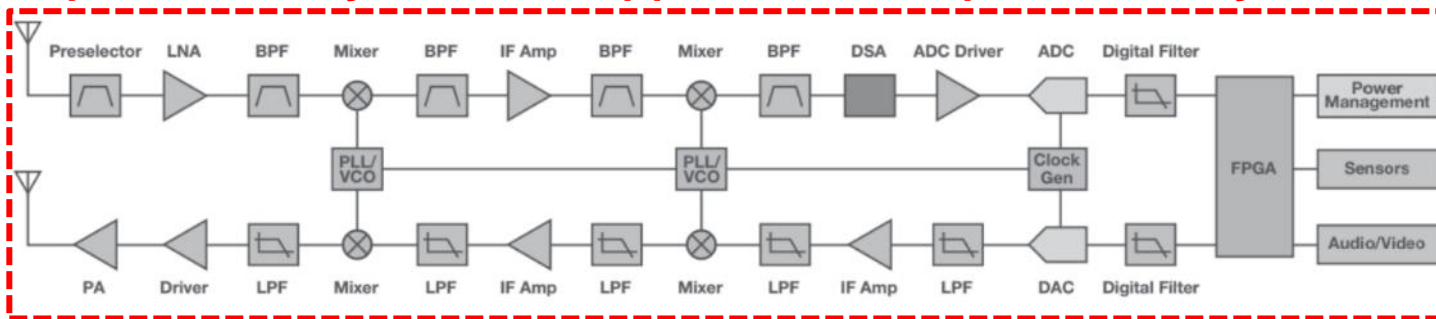
# What is SDR (Software Defined Radio)?

Reconfigurable, multipurpose radio:



Optimized only for a few applications (Super Heterodyne):

(Not SDR)



# What is SDR (Software Defined Radio)?

## Designs are complex

- Multiple skillsets
- Multiple technologies

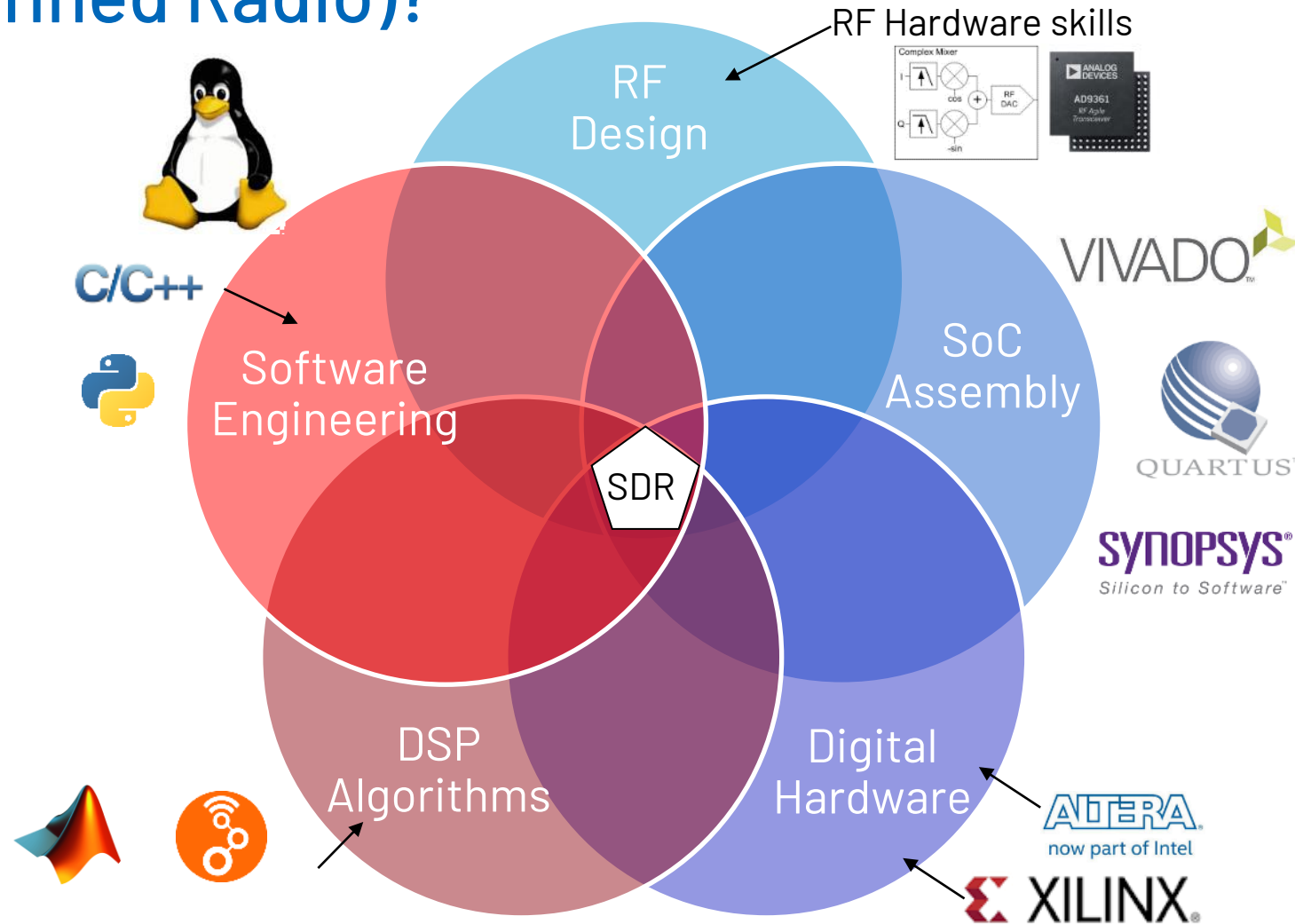
## Radio Developer needs

- Fast prototyping
  - Complete reference designs
    - antenna to MATLAB
    - Streaming
    - Targeting
  - Easy to use prototyping platforms
  - Complete workflows – easy to use toolchains
- Path to production
- Reduced system complexity
  - Hardware
  - Software
  - Mechanics

## Reduced risk

## Ease of use sometimes beats performance

- Many decisions are made by the system engineer in the prototyping stage



Communications theory

$$s[2\ell N + n] = \frac{1}{2N} \sum_{k=0}^{2N-1} p_k[\ell] e^{j(2\pi n k / 2N)},$$



# Hardware

## ADALM-PLUTO



~ 230 \$

### Main Specs:

- **2x Tx and 2x Rx** ports 50 Ohm
- **LO Freq. Range:** 70MHz -> 6GHz
- **BW** : 56MHz
- **Sample rate:** 61.44MSPS; 14 bits
- **Interfaces:** USB2, UART

## AD-JUPITER-EBZ



~ 2997 \$

### Main Specs:

- **2x Tx and 2x Rx** ports 50 Ohm
- **LO Freq. Range:** 30MHz -> 6GHz
- **BW** : 40MHz
- **Sample rate:** 61.44MSPS; 16 bits
- **Interfaces:** USB3, 1Gb Ethernet, Display Port, UART

## ADRV9009 SOM



~ 8955 \$

### Main Specs:

- **4x Tx and 4x Rx** (expandable to 8 TRx)
- **LO Freq. Range:** 75MHz -> 6GHz
- **RX BW:** 200MHz, **TX BW:** 450MHz
- **Interfaces:** USB3, 1Gb Ethernet, Display Port, PCIe 3.0 ,SFP, QSFP, UART

# AD-JUPITER-EBZ

## Digitizer

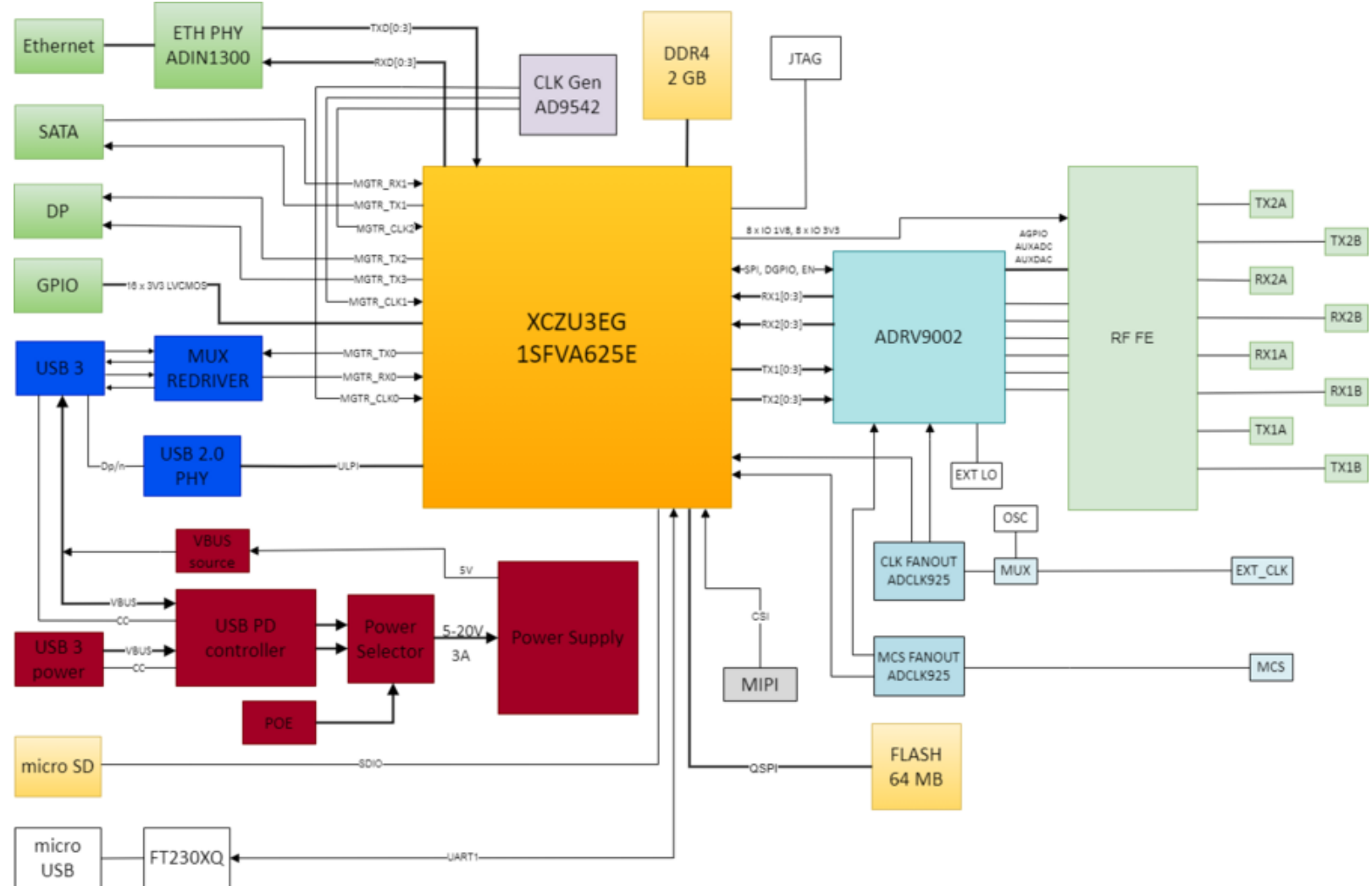
- ADRV9002 RF transceiver
  - 2 RX, 2 TX (A and B)
  - Frequency range 30 MHz to 6 GHz
  - 12 KHz to 40 MHz BW
  - 61.44 MSps maximum sampling rate
- Internal/External Cock selection
- External MCS (multi chip synchronization)
- RF calibration switches
- Customizable RF front end (B channels)

## Processing

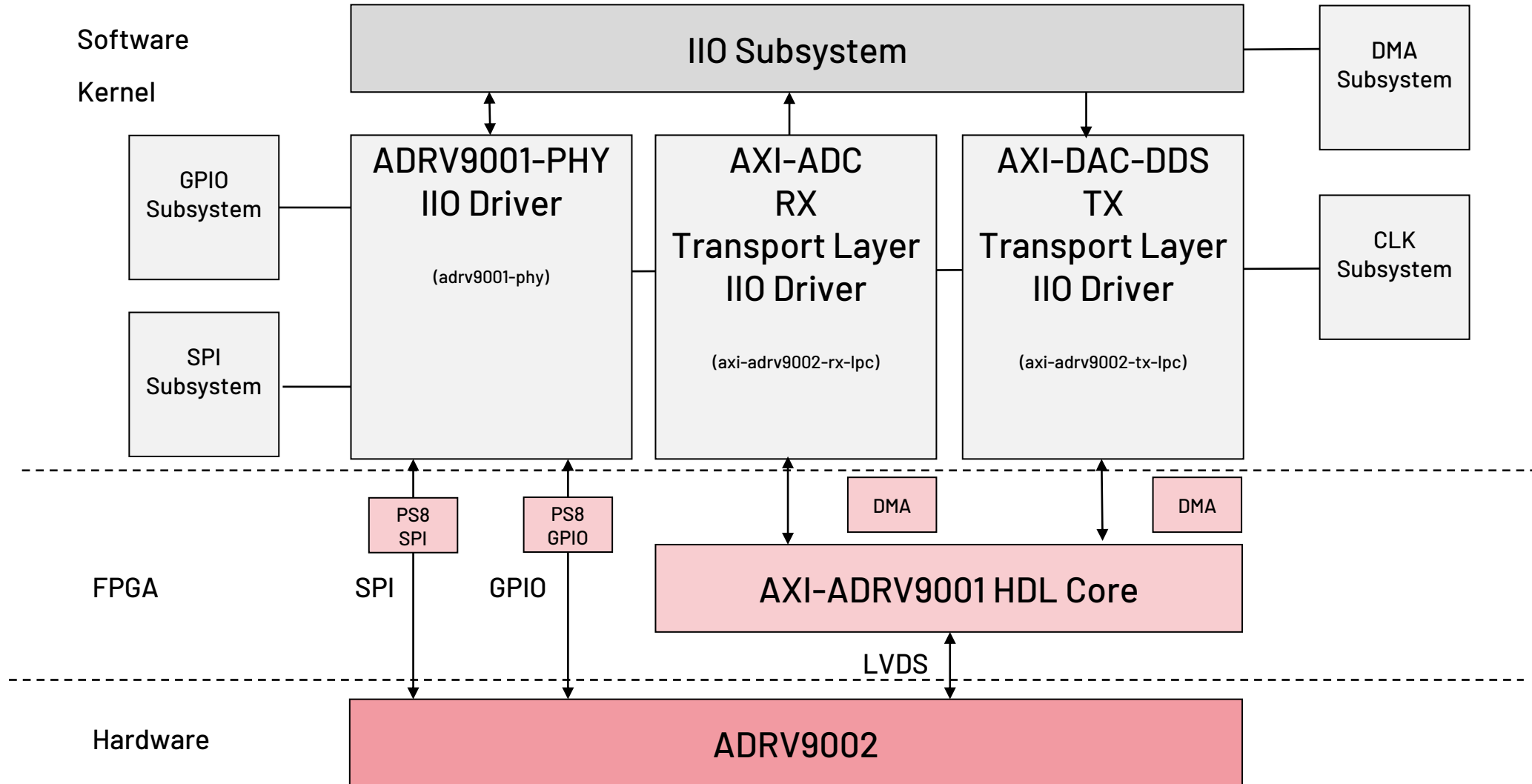
- Zynq UltraScale+ MPSoC XCZU3EG
- DDR4 – 16 Gb (2 GB)
- Boot source
  - FLASH memory 512Mb
  - SD CARD 3.0

## Interfaces

- USB 3.1 Gen 1 (Type C)
- Ethernet 1000BASE-T RGMII
- Display Port v1.2 (1080p)
- SATA 3
- 16 LVTTTL GPIOs



# SW and HDL Architecture



# Setup





# Hands on section

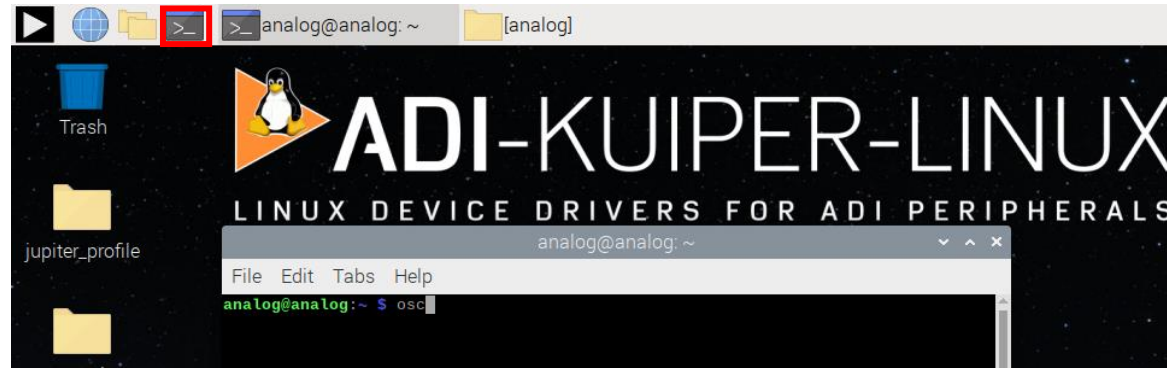
# 1. IIO Oscilloscope

**Transmit and receive a complex sinusoid using the DDS on loopback**

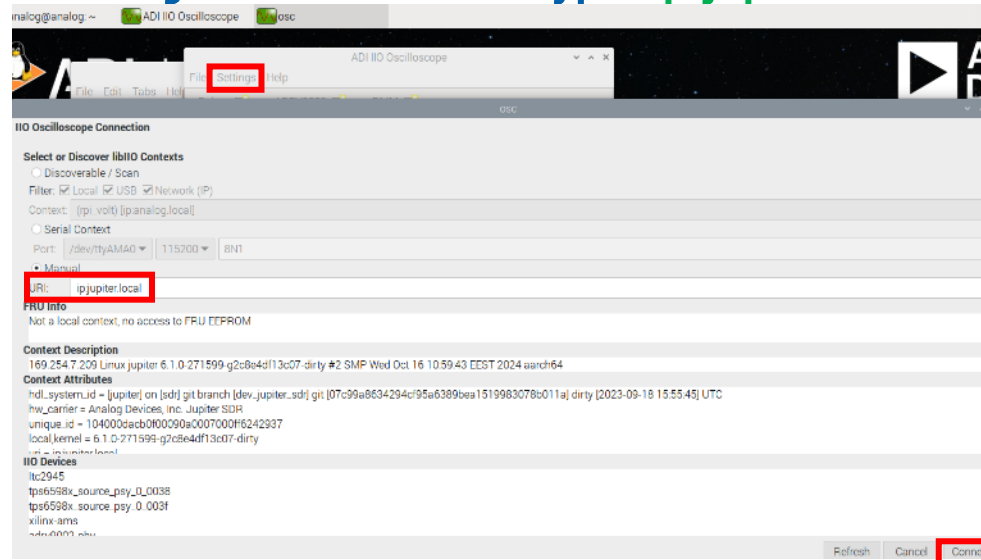


# 1. IIO Oscilloscope

- 1. Open IIO Oscilloscope: Open the terminal from the top bar icon, type “osc” in it and press enter

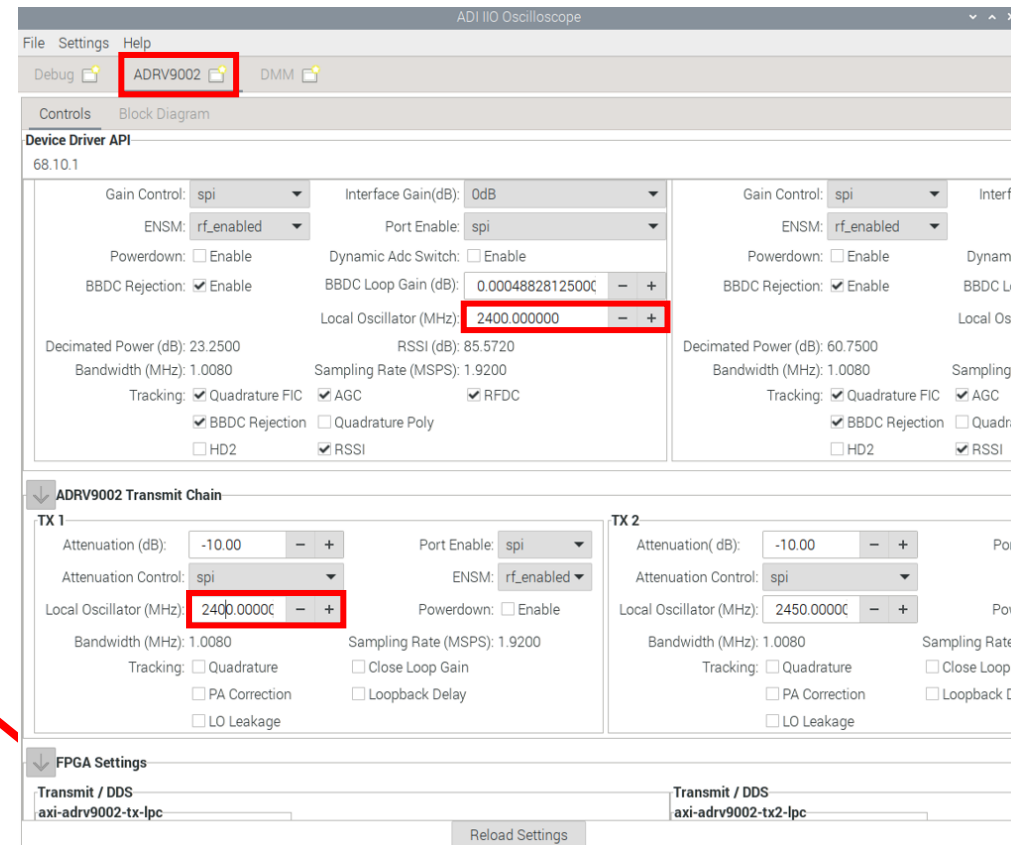
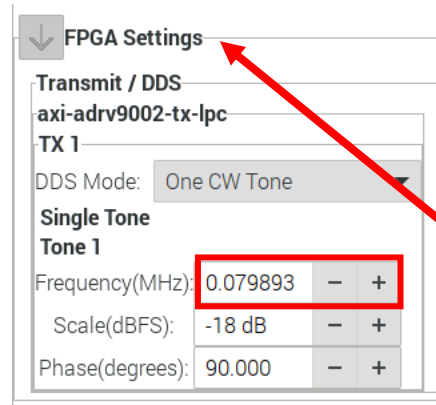


- 2. Connect to board: Go to Settings -> Connect and type “ip:jupiter.local” on the “Manual” section



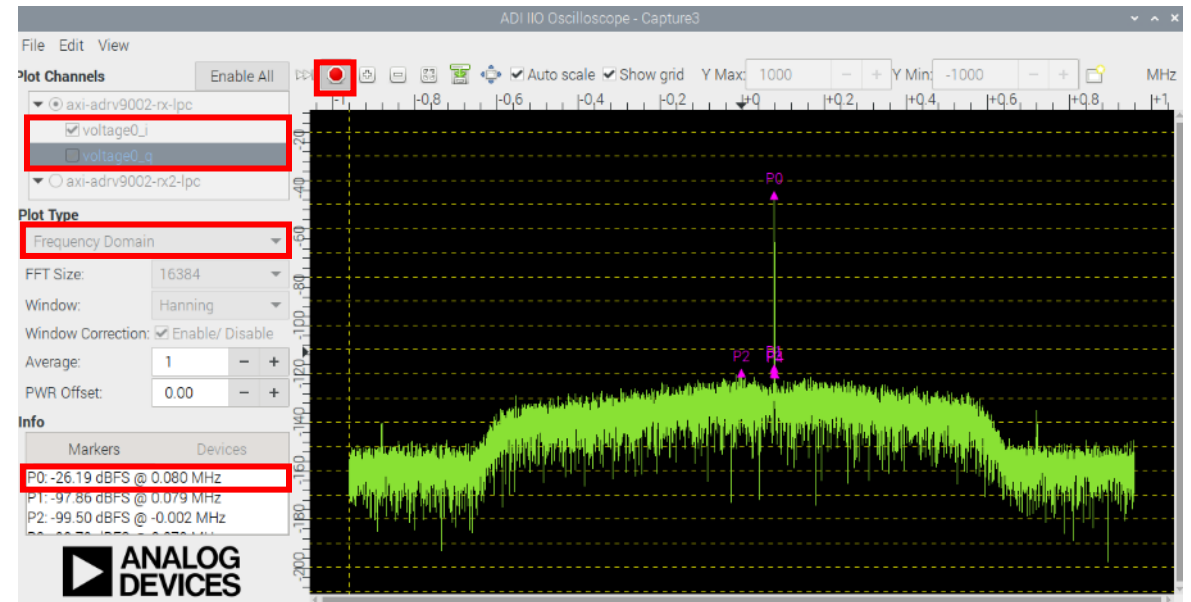
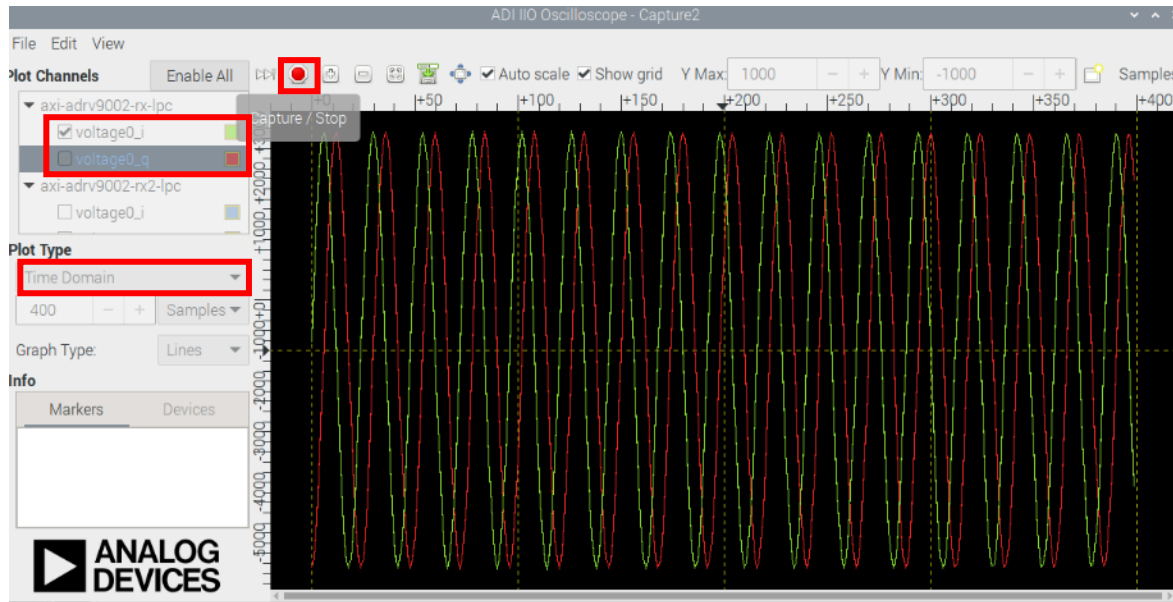
# 1. IIO Oscilloscope

- 3. Use DDS On loopback: We will transmit a 79KHz tone on top of the Tx LO and receive it on the Rx Path. Make sure the Tx LO and Rx LO have the same frequency.

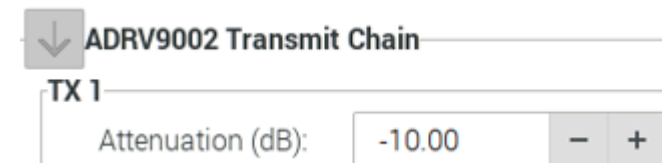


# 1. IIO Oscilloscope

- 4. Open and start the Plot: Go to “File” -> “New Plot”. On the channels window, check both “voltage0\_i” and “voltage0\_q”. If you want to switch between time and frequency domain, you can change the Plot Type. To see peak markers on the Frequency domain, right click on the plot and select “Show Markers”.



- 5. Tweak Tx Attenuation and observe the change in the plot: Decrease or increase the attenuation for Tx1 and observe the change on the FFT Plot. The range for TX is -41dB to 0dB.



# 1. IIO Oscilloscope

## ► 4. Reboot Jupiter:

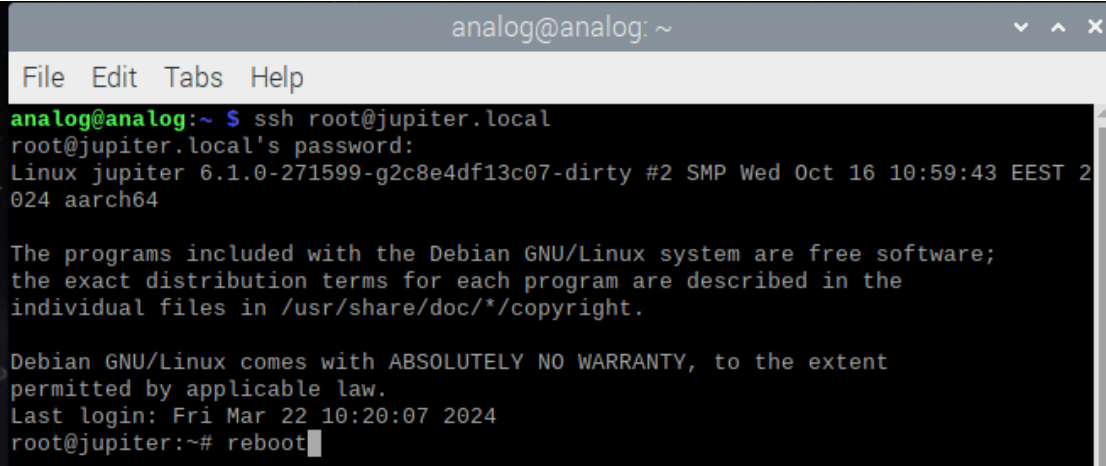
### ► Method 1:

- Open terminal
- Type "`ssh root@jupiter.local`", press Enter and enter "analog" as password
- Type "reboot" in the terminal and press Enter

### ► Method 2:

- Press once the push-button on the back of the Jupiter
- Wait for the LEDs to turn red
- Press once more the push-button to boot again

**! All the instructions you need are also in the "instructions\_ftc2024.pdf" file on your desktop**



```
analog@analog: ~  
File Edit Tabs Help  
analog@analog:~ $ ssh root@jupiter.local  
root@jupiter.local's password:  
Linux jupiter 6.1.0-271599-g2c8e4df13c07-dirty #2 SMP Wed Oct 16 10:59:43 EEST 2024 aarch64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Fri Mar 22 10:20:07 2024  
root@jupiter:~# reboot
```

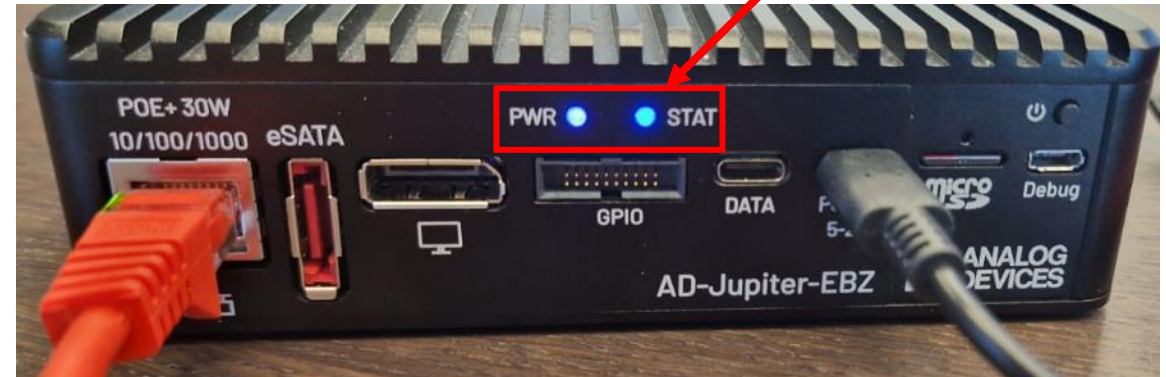
# 1. IIO Oscilloscope

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.



**! All the instructions you need are also in the "instructions\_ftc2024.pdf" file on your desktop**

- Make sure the back panel status blue led is blinking. This shows that the boot stage is successful.



## 2. Transmit and receive a complex sinusoid with GNU Radio



## 2. Transmit and receive a complex sinusoid with GNU Radio

### ► What is GNU Radio?

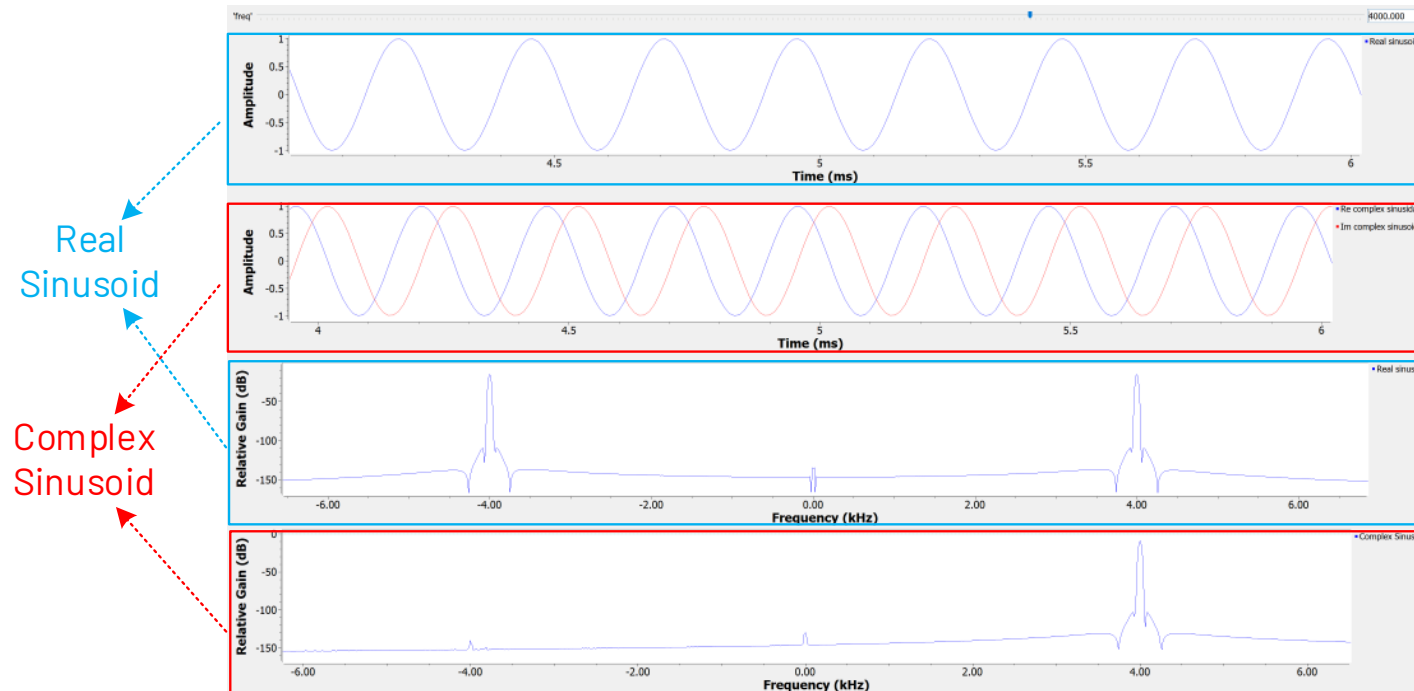


- GNU Radio is a free & open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in research, industry, academia, government, and hobbyist environments to support both wireless communications research and real-world radio systems.

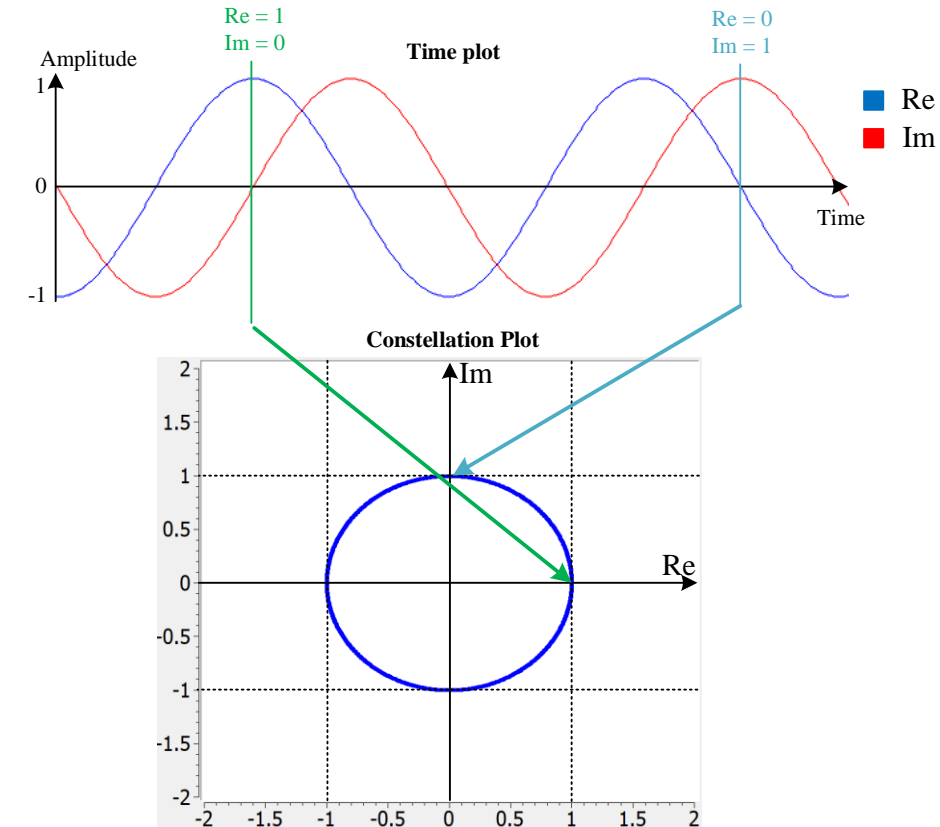
## 2. Transmit and receive a complex sinusoid with GNU Radio

### What is a complex sinusoid?

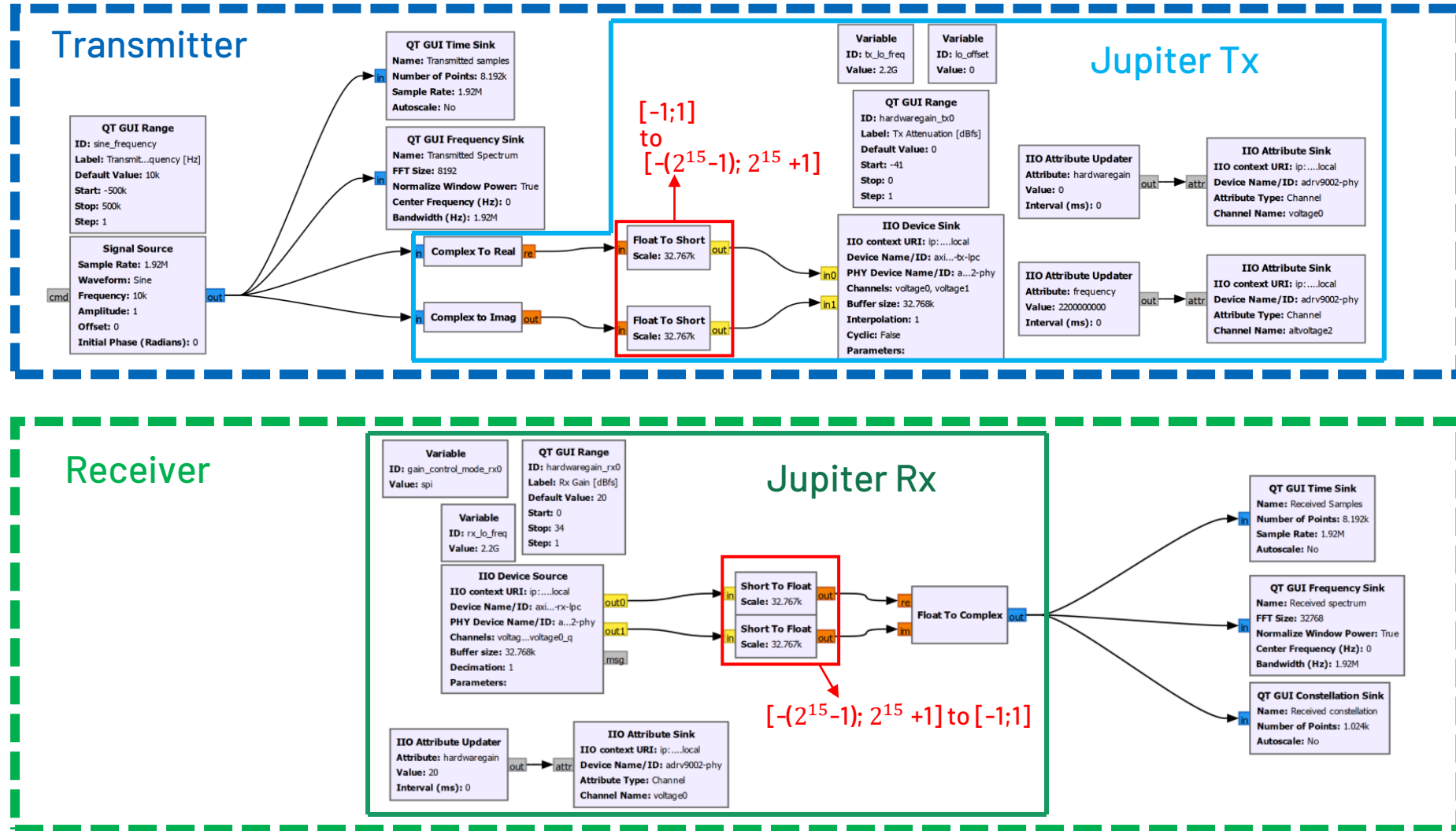
- Real sinusoid:  $e^{j2\pi ft} + e^{-j2\pi ft} = 2 \cos(2\pi ft)$
- Complex sinusoid:  $e^{j2\pi ft} = \underbrace{\cos(2\pi ft)}_{\text{I (In phase)}} + j \underbrace{\sin(2\pi ft)}_{\text{Q (Quadrature phase)}}$



- Complex sinusoid (Constellation Plot):

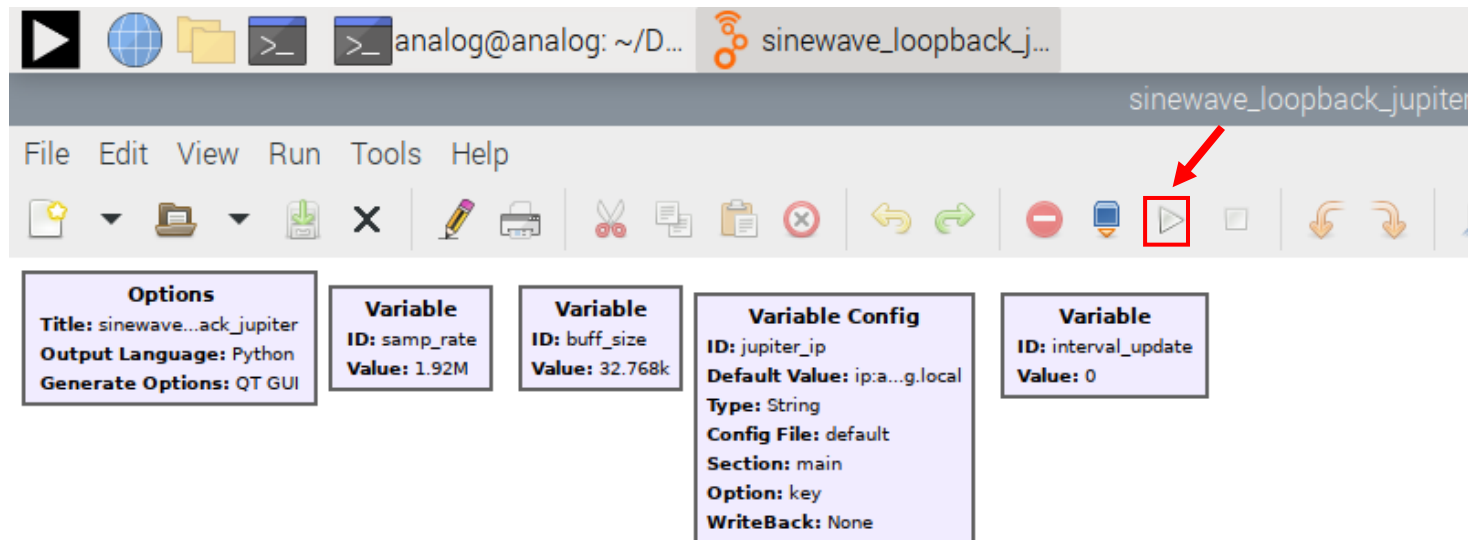


## 2. Transmit and receive a complex sinusoid with GNU Radio

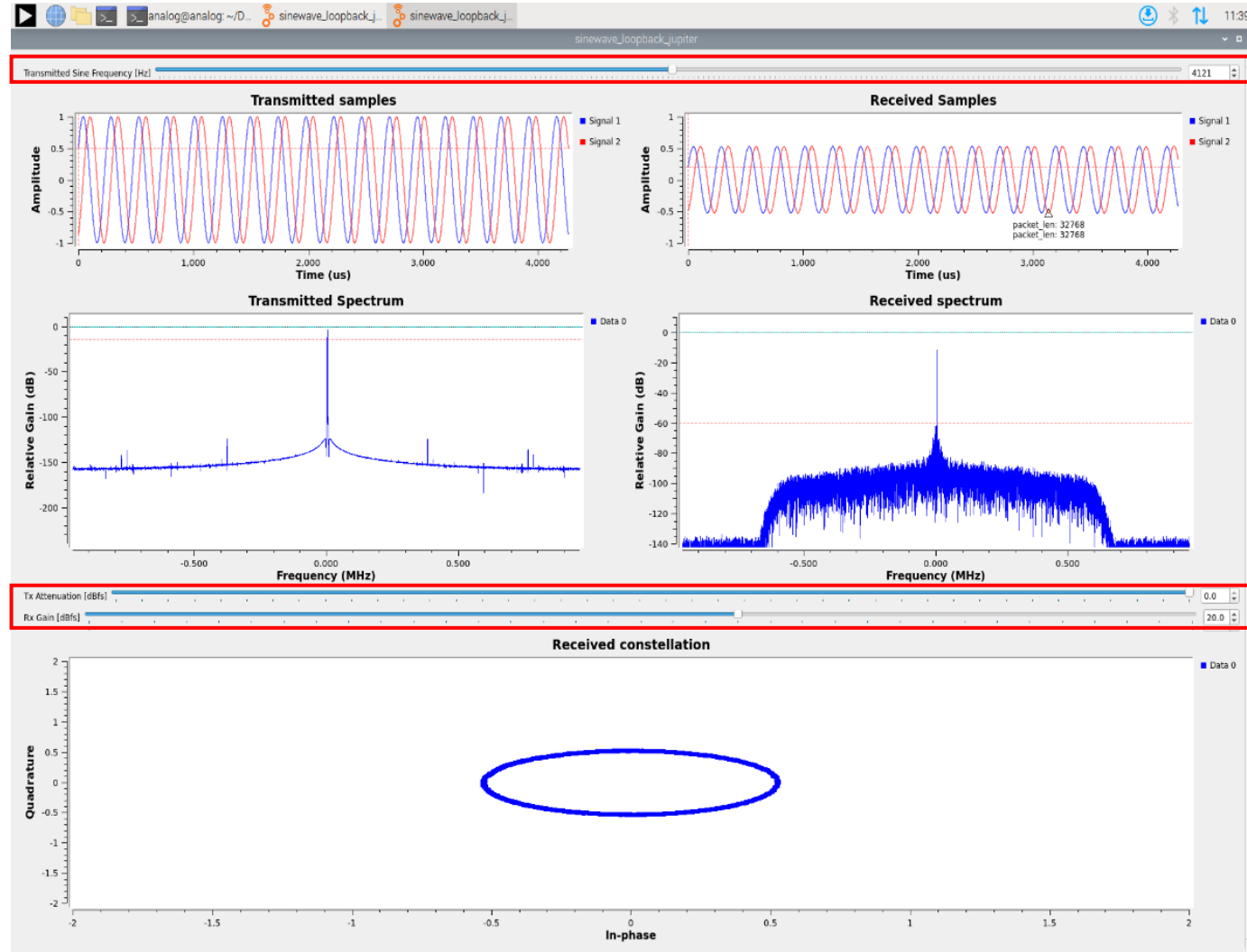


## 2. Transmit and receive a complex sinusoid with GNU Radio

- To run the flowgraph press the arrow from the top bar inside the app as shown below.



## 2. Transmit and receive a complex sinusoid with GNU Radio



► Tweak this Slider to change the frequency of the Tx sinusoid

► Tweak these Sliders to change the gain on Rx and the attenuation on Tx



## 2. Transmit and receive a complex sinusoid with GNU Radio

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

- In GNU Radio companion app, open from File -> Open:

```
/home/analog/Desktop/ftc_2024/1_sinewave_loopback_gnuradio/sinewave_loopback_jupiter.grc
```

# 3. Transmit and receive a complex sinusoid with Python

### 3. Transmit and receive a complex sinusoid with Python

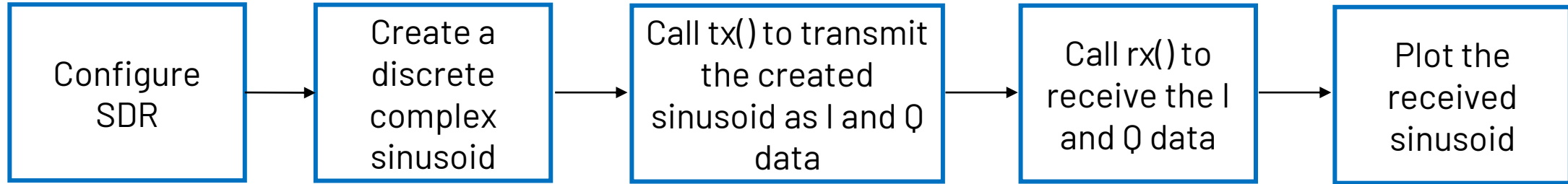
#### ► What is PyADI-IIO?



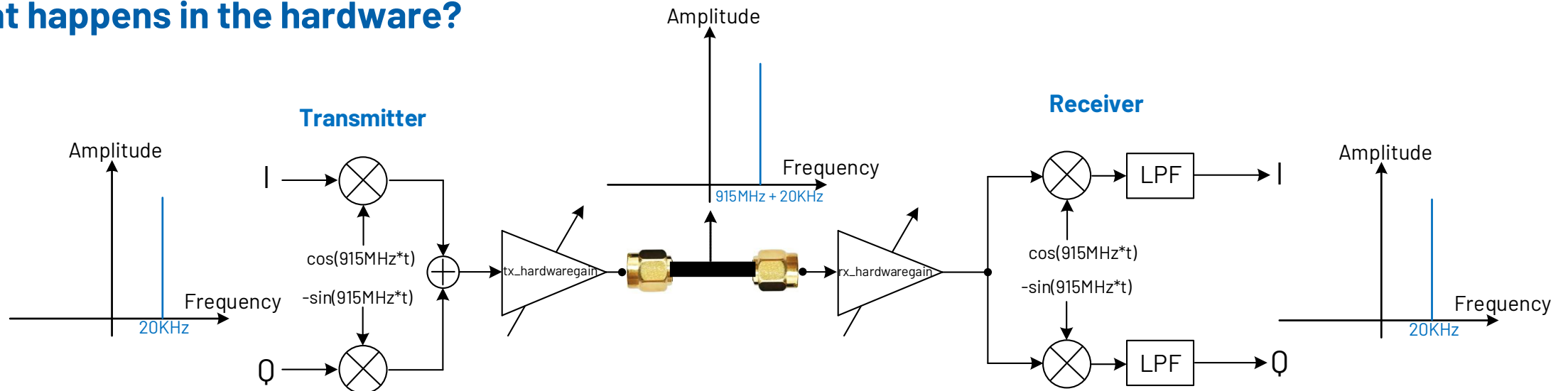
- **PyADI-IIO** is a python abstraction module for ADI hardware with IIO drivers to make them easier to use. The libIIO interface although extremely flexible can be cumbersome to use due to the amount of boilerplate code required for even simple examples, especially when interfacing with buffers. This module has custom interfaces classes for specific parts and development systems which can generally, make them easier to understand and use. To get up and running with a device can be as simple as a few lines of code.

# 3. Transmit and receive a complex sinusoid with Python

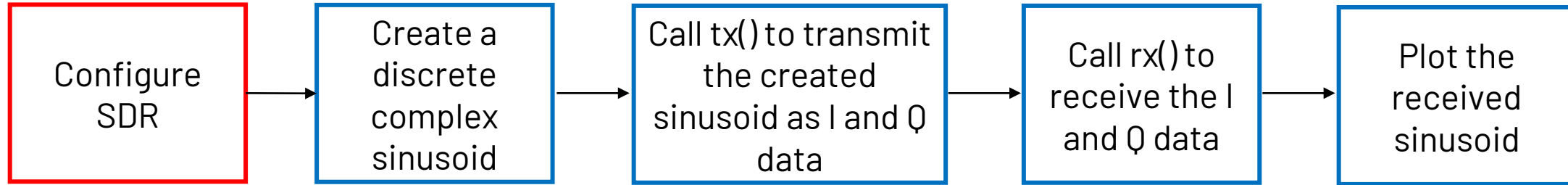
## ► What does this code do?



## ► What happens in the hardware?



# 3. Transmit and receive a complex sinusoid with Python



```

1 import numpy as np
2 import adi
3 import matplotlib.pyplot as plt
4 import time
5
6 #####
7 # Configure SDR #####
8 #####
9 sdr = adi.adrv9002(uri="ip:jupiter.local") # Create Radio
10
11 frequency_tx1 = 15000 # 15 kHz sinewave to be transmitted
12 frequency_tx2 = frequency_tx1 * 5 # 75 kHz
13 num_periods_tx1 = 50
14 num_periods_tx2 = num_periods_tx1 * 5
15 amplitude = 2**14 # maximum is (2**15)
16 center_freq_tx1_rx1 = 2200000000 # Hz
17 center_freq_tx2_rx2 = 3200000000 # Hz
18 sample_rate_tx1 = sdr.tx0_sample_rate
19 sample_rate_tx2 = sdr.tx1_sample_rate
20 print("sample rate ch Tx1: ", sample_rate_tx1)
21 print("sample rate ch Tx2: ", sample_rate_tx2)
22 sample_rate_rx1 = sdr.rx0_sample_rate
23 sample_rate_rx2 = sdr.rx1_sample_rate
24 print("sample rate ch Rx1: ", sample_rate_rx1)
25 print("sample rate ch Rx2: ", sample_rate_rx2)
26 num_samps_tx1 = int((num_periods_tx1*sample_rate_rx1)/frequency_tx1) # number of samples per call to tx() and
27 num_samps_tx2 = int((num_periods_tx2*sample_rate_rx1)/frequency_tx1) # number of samples per call to tx() and
28 num_samps_rx1 = num_samps_tx1
29 num_samps_rx2 = num_samps_tx1
  
```

Change LO frequency for:  
Tx1, Rx1, Tx2, Rx2

```

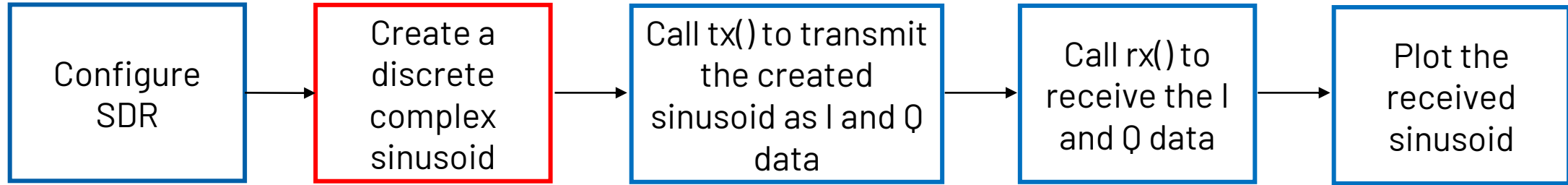
30
31 # Settings Rx1 Tx1
32 sdr.rx_ensm_mode_chan0 = "rf_enabled"
33 sdr.tx_hardwaregain_chan0 = 0
34 sdr.rx_hardwaregain_chan0 = 15
35 sdr.tx_cyclic_buffer = True
36 sdr.tx_buffer_size = num_samps_tx1 # number of samples per call to tx()
37 sdr.rx_buffer_size = num_samps_rx1
38 sdr.tx0_lo = center_freq_tx1_rx1
39 sdr.rx0_lo = center_freq_tx1_rx1
40
41 # Settings Rx2 Tx2
42 sdr.rx_ensm_mode_chan1 = "rf_enabled"
43 sdr.tx_hardwaregain_chan1 = 0
44 sdr.rx_hardwaregain_chan1 = 15
45 sdr.tx2_cyclic_buffer = True
46 sdr.tx2_buffer_size = num_samps_tx2 # number of samples per call to tx()
47 sdr.rx2_buffer_size = num_samps_rx2
48 sdr.tx1_lo = center_freq_tx2_rx2
49 sdr.rx1_lo = center_freq_tx2_rx2
50 #####
51 #####
  
```

Other Settings  
for Rx1 and Tx1

Other Settings  
for Rx2 and Tx2



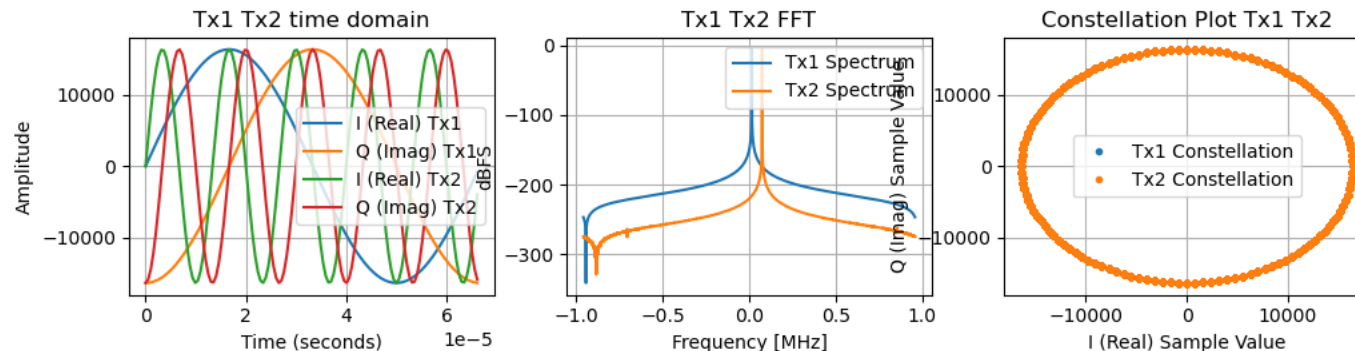
# 3. Transmit and receive a complex sinusoid with Python



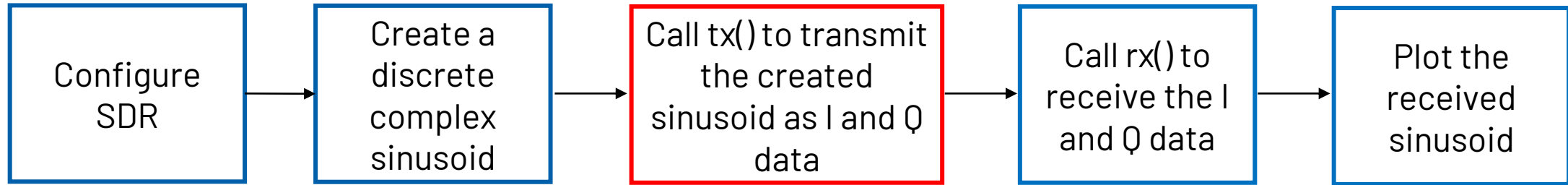
$$e^{j2\pi ft} = \cos(2\pi ft + \varphi) + j \sin(2\pi ft + \varphi)$$

```

53 #####
54 # Create and plot a complex sinusoid #####
55 #####
56 # Calculate time values for Tx1
57 t1 = np.arange(num_samps_tx1) / sample_rate_tx1
58 # Generate sinusoidal waveform
59 phase_shift = -np.pi/2 # Shift by -90 degrees
60 tx1_samples = amplitude * (np.cos(2 * np.pi * frequency_tx1 * t1 + phase_shift) + 1j*np.sin(2 * np.pi * frequency_tx1 * t1 + phase_s
61
62 # Calculate time values for Tx2
63 t2 = np.arange(num_samps_tx2) / sample_rate_tx2
64 # Generate sinusoidal waveform
65 tx2_samples = amplitude * (np.cos(2 * np.pi * frequency_tx2 * t2 + phase_shift) + 1j*np.sin(2 * np.pi * frequency_tx2 * t2 + phase_s
66
  
```



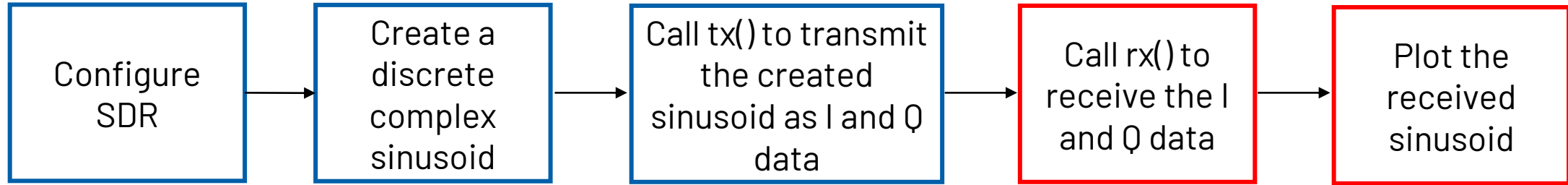
### 3. Transmit and receive a complex sinusoid with Python



► **Using PyADI-II0, only these functions you have to call to start transmitting:**

```
122 sdr.tx(tx1_samples) # start transmitting on Tx1
123 sdr.tx2(tx2_samples) # start transmitting on Tx2
```

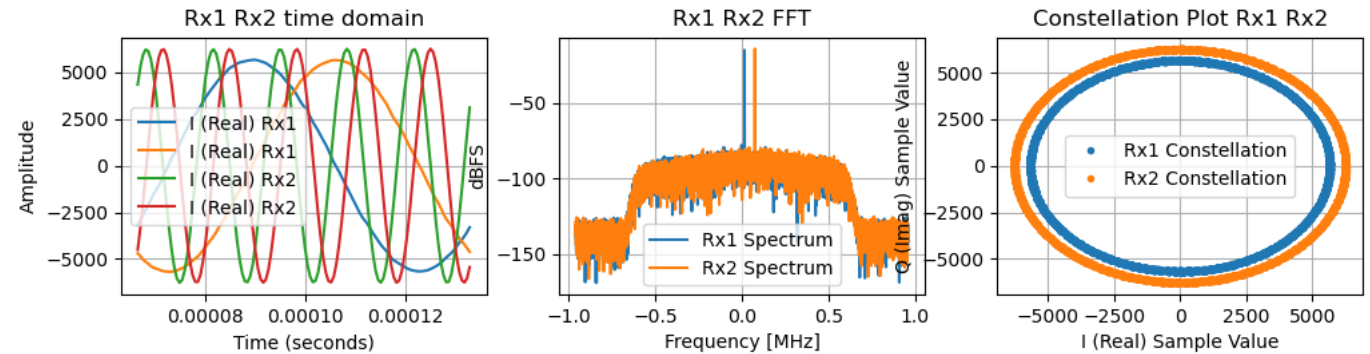
# 3. Transmit and receive a complex sinusoid with Python



```

134 # Call Rx function to receive transmission and plot the data#####
135 #####
136 # Receive samples on Rx1 and Rx2
137 rx1_samples = sdr.rx()
138 rx2_samples = sdr.rx2()
139
140 # Stop transmitting
141 sdr.tx_destroy_buffer()
142 sdr.tx2_destroy_buffer()
143
144 # Time values for Rx1
145 t1 = np.arange(num_samps_rx1) / sample_rate_rx1
146 # Time values for Rx1
147 t2 = np.arange(num_samps_rx2) / sample_rate_rx2
148
149 if(frequency_tx1 < frequency_tx2):
150     plotted_num_samples_time_rx = int(num_samps_rx1/num_periods_tx1)
151 else:
152     plotted_num_samples_time_rx = int(num_samps_rx2/num_periods_tx2)
153
154 # Plot Rx1 and Rx2 time domain
155 axs[1,0].plot(t1[plotted_num_samples_time_rx:plotted_num_samples_time_rx*2], ...,
156 axs[1,0].plot(t1[plotted_num_samples_time_rx:plotted_num_samples_time_rx*2], np.
157 axs[1,0].plot(t2[plotted_num_samples_time_rx:plotted_num_samples_time_rx*2], np.
158 axs[1,0].plot(t2[plotted_num_samples_time_rx:plotted_num_samples_time_rx*2], np.
159 axs[1,0].grid(True)
160 axs[1,0].legend()
161 axs[1,0].set_title('Rx1 Rx2 time domain')
162 axs[1,0].set_xlabel('Time (seconds)')
163 axs[1,0].set_ylabel('Amplitude')
164
  
```

Only these two functions needs to be called in order to receive.



# 3. Transmit and receive a complex sinusoid with Python

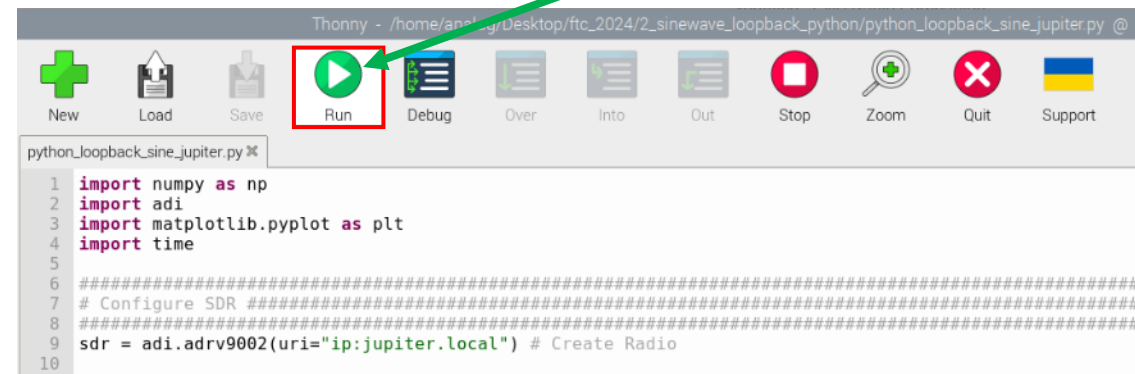
- For this example, use two SMA cables to make **loopback** between TX1 -> RX1 and TX2 -> RX2 as depicted below:



- Open "**python\_loopback\_sine\_jupiter.py**" file using "Thonny" IDE from:

**"/home/analog/Desktop/ftc\_2024/2\_sinewave\_loopback\_python/"**

- To run the python script, click on the "**Run**" button:



## 3. Transmit and receive a complex sinusoid with Python

► Change the profile from command line on Jupiter to the one with 61.44 MHz Sample Rate and run the python example again:

**Commands** to run in the termina (! You can **copy and paste commands** from **"terminal\_commands.txt"**, which is on Desktop):

# First connect to Jupiter with ssh:

\$ **ssh root@jupiter.local**

# enter "analog" as password

\$ **cat /home/analog/workspace/jupiter\_profiles/jupiter\_61\_44MHz\_profile.json > /sys/bus/iio/devices/iio\:device1/profile\_config**

# exit ssh connection

\$ **exit**

# run again the python code (from Thonny or command line)

\$ **python3 /home/analog/Desktop/ftc\_2024/2\_sinewave\_loopback\_python/python\_loopback\_sine\_jupiter.py**

**You should see the same plots but with more samples and a wider spectrum.**

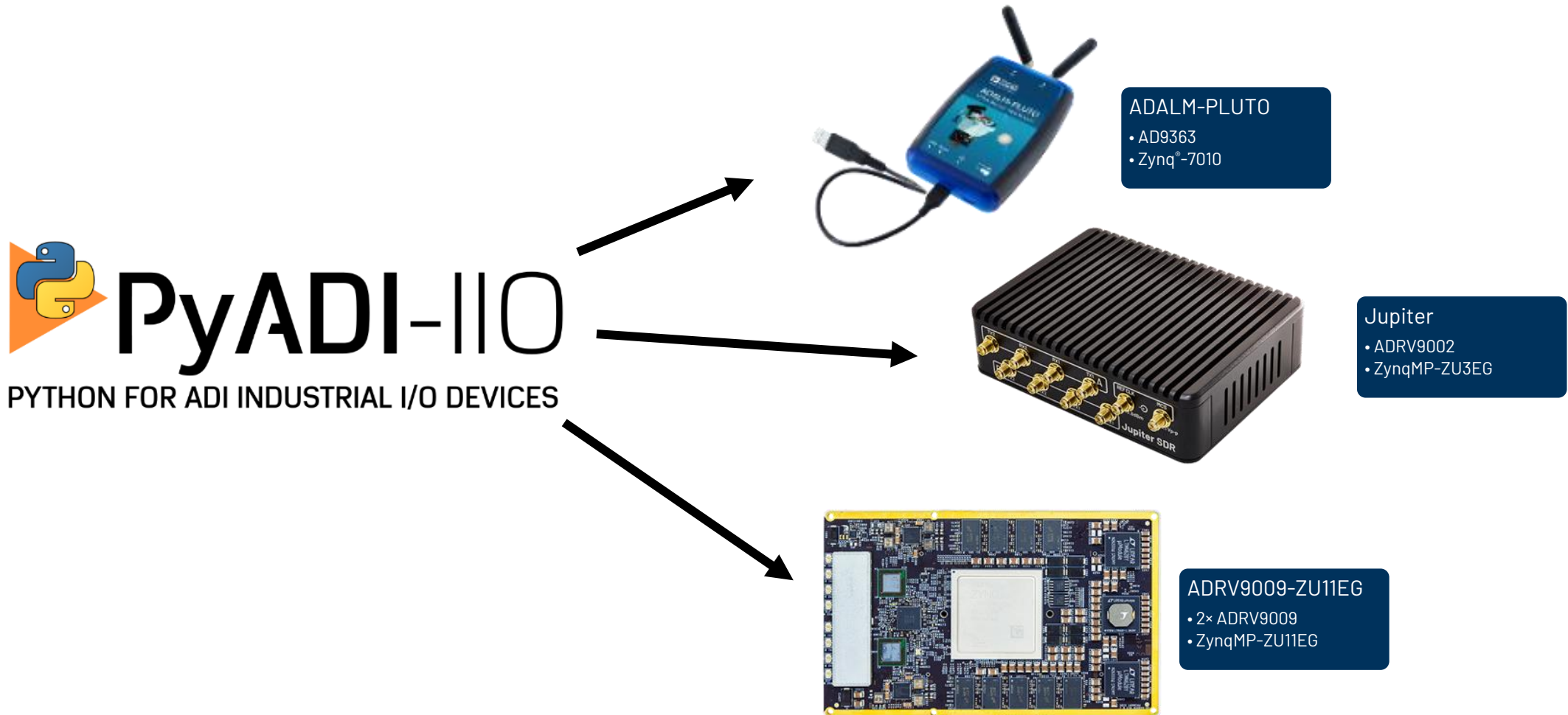


## 3. Transmit and receive a complex sinusoid with Python

- ▶ After you finished playing with this example, **reboot Jupiter**:
  - ▶ Method 1:
    - ▶ Open terminal and run the following **commands**:
    - ▶ **\$ ssh root@jupiter.local**
    - ▶ **# enter "analog" as password**
    - ▶ **\$ reboot**
  - ▶ Method 2
    - ▶ Press once the push-button on the back of the Jupiter
    - ▶ Wait for the LEDs to turn **red**
    - ▶ Press once more the push-button to boot again

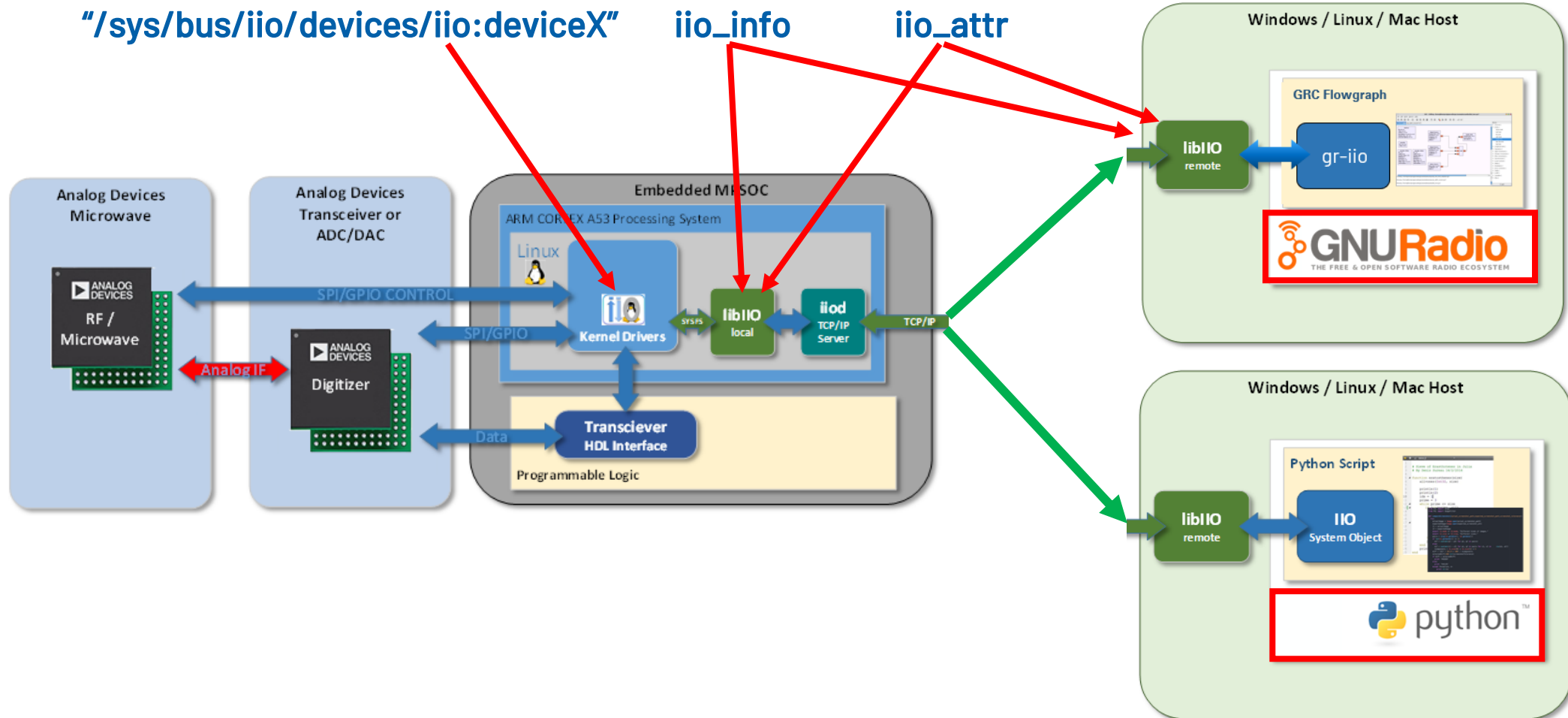
# 3. Transmit and receive a complex sinusoid with Python

- Using PyADI-II0 you can reuse the code and run it on different SDRs like Pluto or Talise.

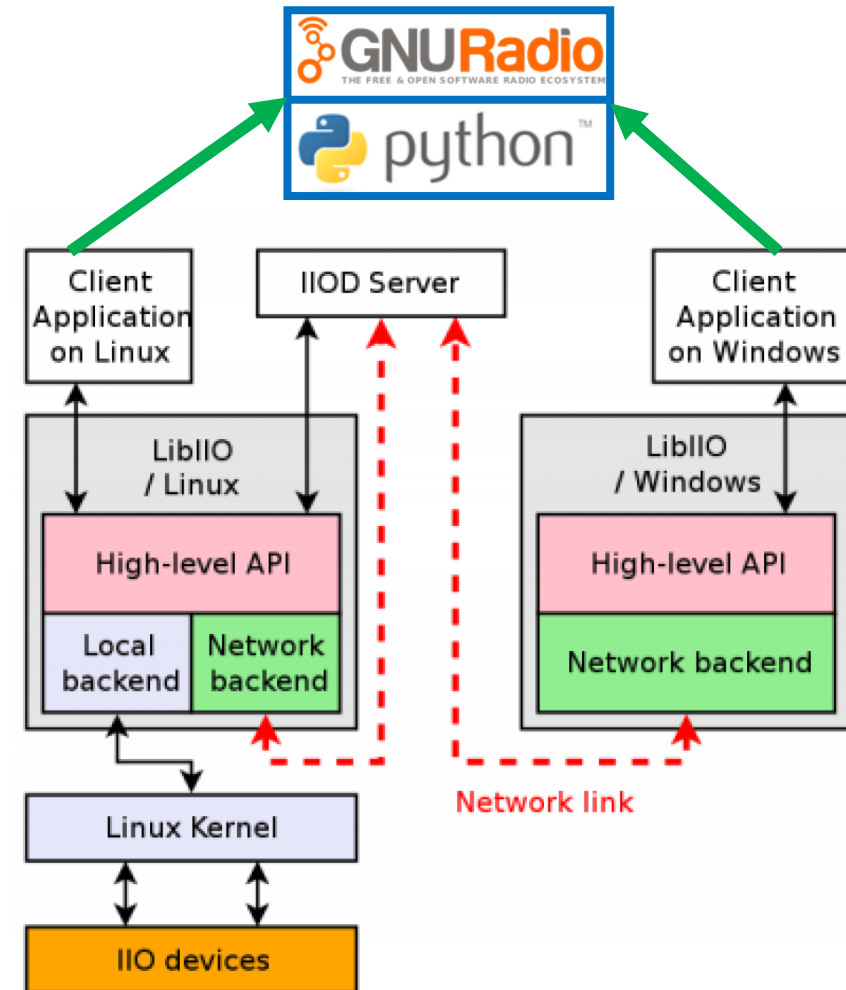
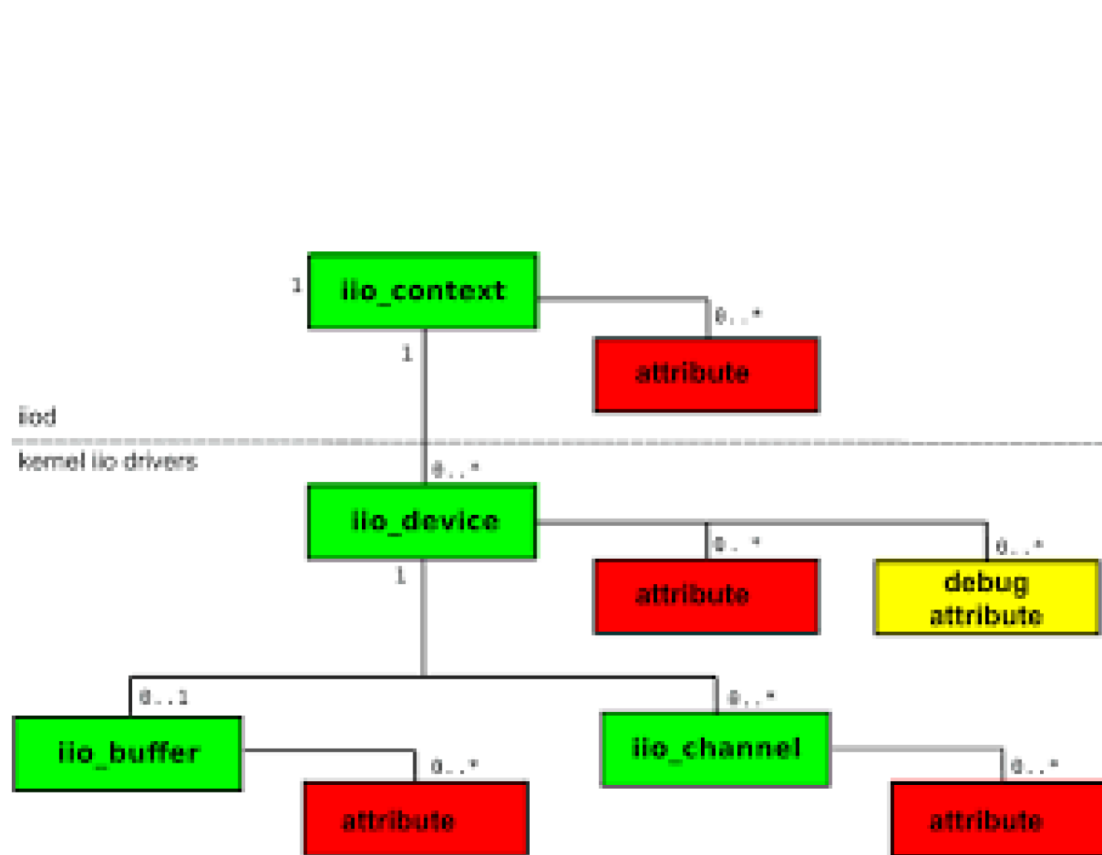


# 4. How to explore and tweak device attributes

# 4. How to explore and tweak device attributes



## 4. How to explore and tweak device attributes





## 4. How to explore and tweak device attributes

### ► What devices are on Jupiter SDR from a Software perspective?

To see all devices, run the following command in the terminal (not necessarily on the target):

**\$ iio\_attr -u ip:jupiter.local -d**

```
analog@analog:~ $ iio_attr -u ip:jupiter.local -d
IIO context has 10 devices:
  hwmon0, ltc2945: found 0 device attributes
  hwmon1, tps6598x_source_psy_0_0038: found 0 device attributes
  hwmon2, tps6598x_source_psy_0_003f: found 0 device attributes
  iio:device0, xilinx-ams: found 1 device attributes
  iio:device1, adrv9002-phy: found 17 device attributes
  iio:device2, axi-adrv9002-rx-lpc: found 3 device attributes
  iio:device3, axi-adrv9002-rx2-lpc: found 1 device attributes
  iio:device4, axi-adrv9002-tx-lpc: found 3 device attributes
  iio:device5, axi-adrv9002-tx2-lpc: found 3 device attributes
  iio_sysfs_trigger: found 2 device attributes
```

**'adrv9002-phy' -> contains most of the configuration attributes (hardwaregain, L0 freq. etc.)**

**'axi-adrv9002-rx-lpc' -> ADC on the RX1 path**

**'axi-adrv9002-rx2-lpc' -> ADC on the RX2 path**

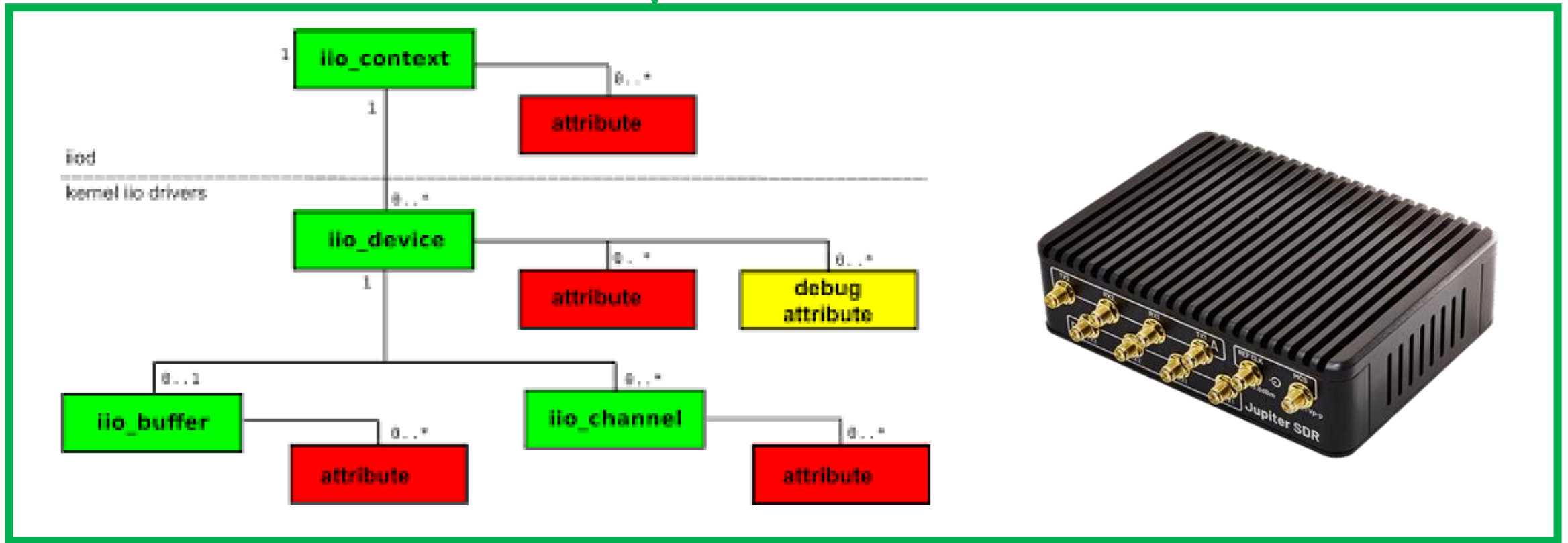
**'axi-adrv9002-tx-lpc' -> DAC on the TX1 path**

**'axi-adrv9002-tx2-lpc' -> DAC on the TX2 path**

**Most Important IIO Devices used in  
the examples presented in this  
workshop**

## 4. How to explore and tweak device attributes

- Use `iio_info` command to display the whole context.



## 4. How to explore and tweak device attributes

- How to change an attribute from command line (e.g. for hardwaregain of RX1)?

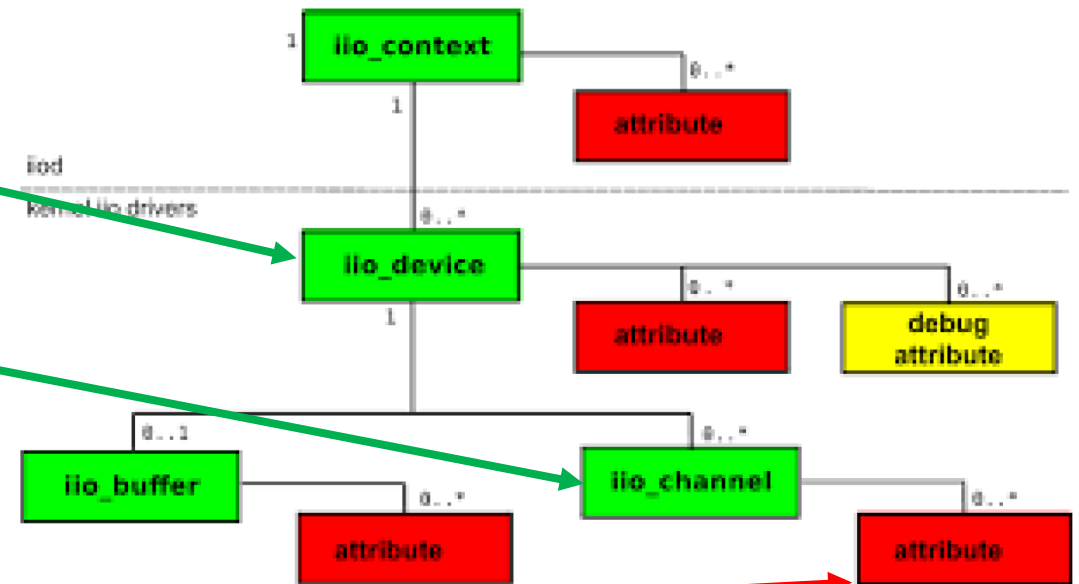
Find the **channel name** and **channel attribute name**

\$ iio\_info | grep -A 200 **adrv9002-phy**

voltage0: (input)

32 channel-specific attributes found:

```
attr 0: agc_tracking_en value: 1
attr 1: bbdc_loop_gain_raw value: 1048576
attr 2: bbdc_rejection_en value: 1
attr 3: bbdc_rejection_tracking_en value: 1
attr 4: decimated_power value: 16.500 dB
attr 5: digital_gain_control_mode value: spi
attr 6: digital_gain_control_mode_available value: automatic spi
attr 7: dynamic_adc_switch_en value: 0
attr 8: en value: 1
attr 9: ensm_mode value: rf_enabled
attr 10: ensm_mode_available value: calibrated primed rf_enabled
attr 11: gain_control_mode value: spi
attr 12: gain_control_mode_available value: spi pin automatic
attr 13: hardwaregain value: 34.000000 dB
attr 14: hd_tracking_en value: 0
attr 15: interface_gain value: 0dB
attr 16: interface_gain_available value: 0dB
attr 17: nco_frequency ERROR: Unknown error 524
attr 18: orx_bbdc_rejection_en ERROR: No such device (19)
```



## 4. How to explore and tweak device attributes

- How to change an attribute from command line (e.g. for hardwaregain of RX1)?

**\$ iio\_attr -u ip:jupiter.local -c -i adrv9002-phy voltage0 hardwaregain5**

```
analog@imhotep:~$ iio_attr -h
Usage
  iio_attr [OPTION]... -d [device] [attr] [value]
                    -c [device] [channel] [attr] [value]
                    -B [device] [attr] [value]
                    -D [device] [attr] [value]
                    -C [attr]

Options
  -h, --help           : Show this help and quit.
  -I, --ignore-case    : Ignore case distinctions.
  -q, --quiet          : Return result only.
  -a, --auto           : Use the first context found.

Optional qualifiers
  -u, --uri            : Use the context at the provided URI.
  -i, --input-channel  : Filter Input Channels only.
  -o, --output-channel : Filter Output Channels only.

Attribute types
  -s, --scan-channel   : Filter Scan Channels only.
  -d, --device-attr    : Read/Write device attributes
  -c, --channel-attr   : Read/Write channel attributes.
  -C, --context-attr   : Read IIIO context attributes.
  -B, --buffer-attr    : Read/Write buffer attributes.
  -D, --debug-attr     : Read/Write debug attributes.
```

# 4. How to explore and tweak device attributes

## ► Some methods to change attributes:

### ► pyadi-iio

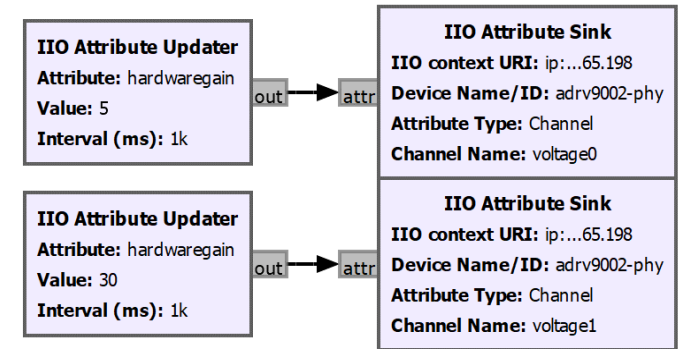
```
1 import numpy as np
2 import adi
3 import matplotlib.pyplot as plt
4 import time
5
6 #####
7 # Configure SDR #####
8 #####
9 sdr = adi.adrv9002(uri="ip:jupiter.local") # Create Radio
10
34 sdr.rx_hardwaregain_chan0 = 15
```

### ► iio\_attr

`$ iio_attr -u ip:jupiter.local -c -i adrv9002-phy 'voltage0' hardwaregain 5`

► [https://wiki.analog.com/resources/tools-software/linux-software/libiio/iio\\_attr](https://wiki.analog.com/resources/tools-software/linux-software/libiio/iio_attr)

### ► GNU Radio



## ► from “/sys/bus/iio/devices/iio:deviceX” on the target

```
# to write 5 in hardwaregain:
$ echo 5 > /sys/bus/iio/devices/iio:device1/in_voltage0_hardwaregain
# to read hardwaregain:
$ cat /sys/bus/iio/devices/iio:device1/in_voltage0_hardwaregain
```

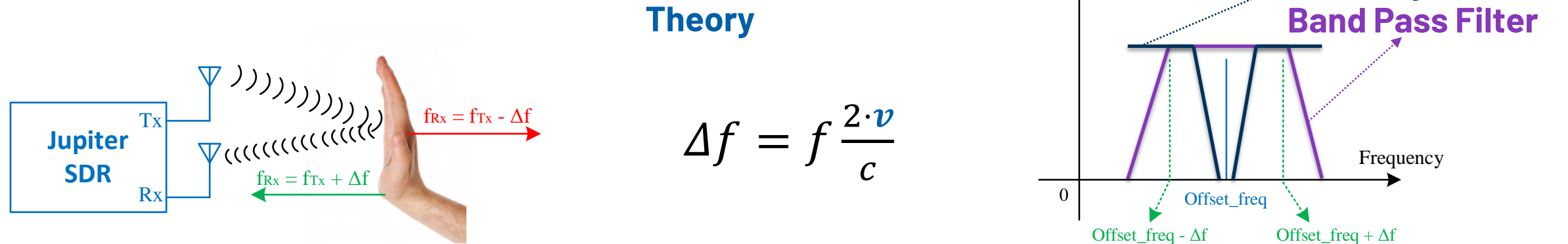
## 4. How to explore and tweak device attributes

- ▶ After you finished playing with this example, **reboot Jupiter**:
  - ▶ Method 1:
    - ▶ Open terminal and run the following **commands**:
    - ▶ **\$ ssh root@jupiter.local**
    - ▶ **# enter "analog" as password**
    - ▶ **\$ reboot**
  - ▶ Method 2
    - ▶ Press once the push-button on the back of the Jupiter
    - ▶ Wait for the LEDs to turn **red**
    - ▶ Press once more the push-button to boot again

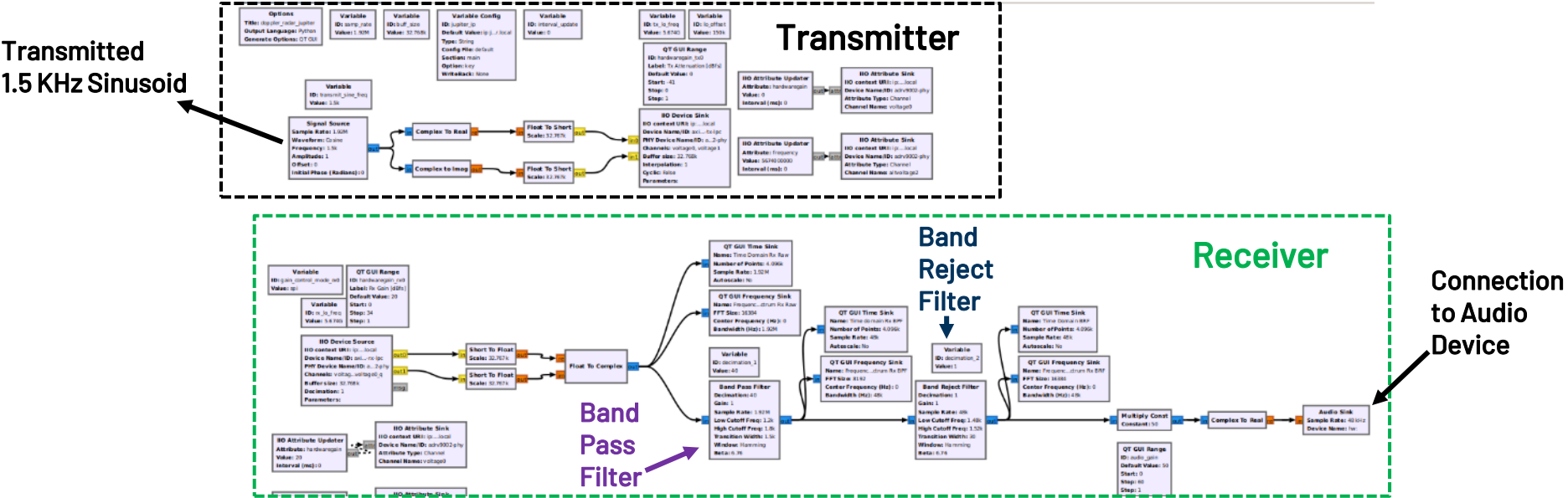


# 5. Doppler Radar with GNU Radio

# 5. Doppler Radar with GNU Radio



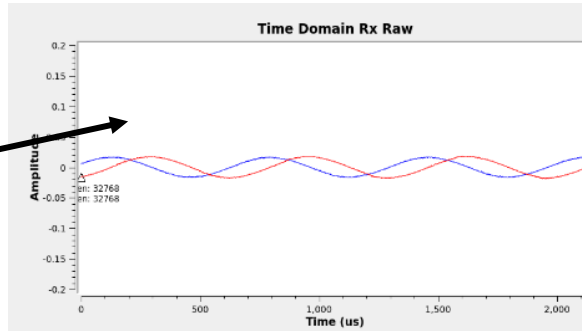
## Implementation



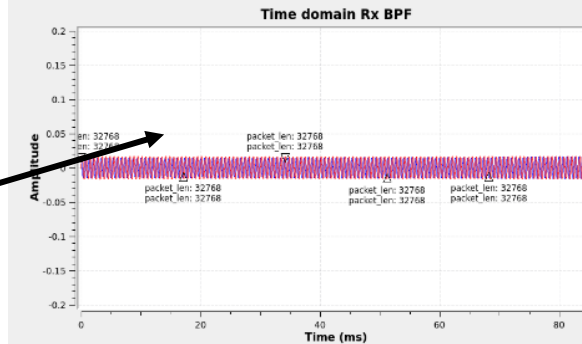
# 5. Doppler Radar with GNU Radio

- After Running the flowgraph, you should hear on the headphones the movement of your hand in front of the antennas

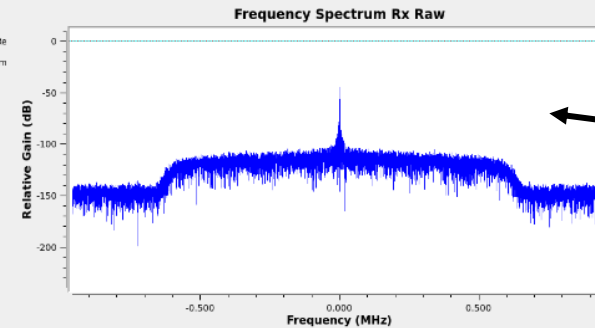
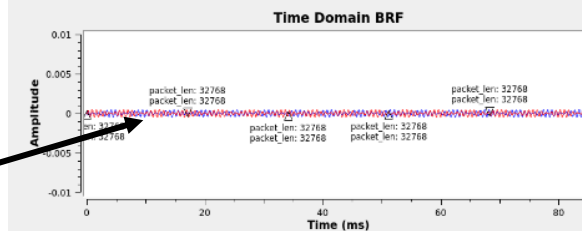
Receive Signal Raw  
– Time domain



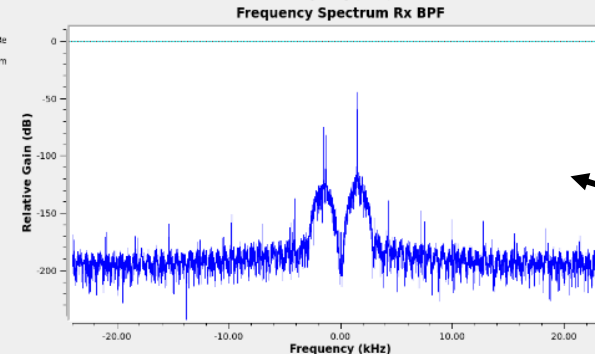
Received Signal after BPF  
– Time domain



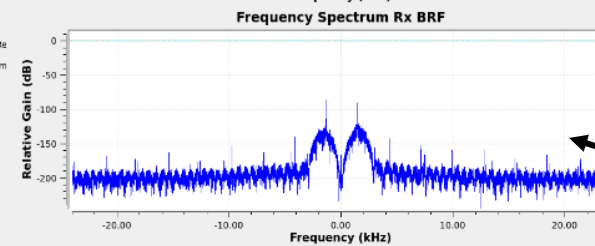
Received Signal after BRF  
– Time domain



Receive Signal Raw  
– Frequency domain



Received Signal after BPF  
– Frequency domain



Received Signal after BRF  
– Frequency domain

## 5. Doppler Radar with GNU Radio

- ▶ Connect one antenna to RX1 and one to TX1 on Jupiter female SMA ports as depicted in the picture below



- ▶ To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

- ▶ In GNU Radio companion app, open from File -> Open:  
`/home/analog/Desktop/ftc_2024/3_doppler_radar_gnudio/doppler_radar_jupiter.grc`

- ▶ Use the given USB Headphones (plug them into the RPI)

## 5. Doppler Radar with GNU Radio

### ► Conclusions:

- This technique (plus some calculations) can be used to detect the velocity of moving objects in radar

### Applications

- The change in frequency can be mapped to speed and direction of movement of the Receiver relative to the Transmitter
- If this change in frequency when the Receiver is moving relative to the transceiver affects the data received, methods to correct this change are required.

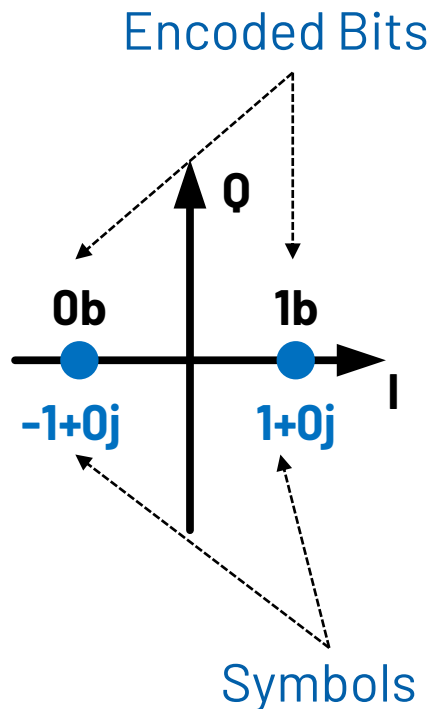
# 6. Phase Shift Keying – BPSK in Python



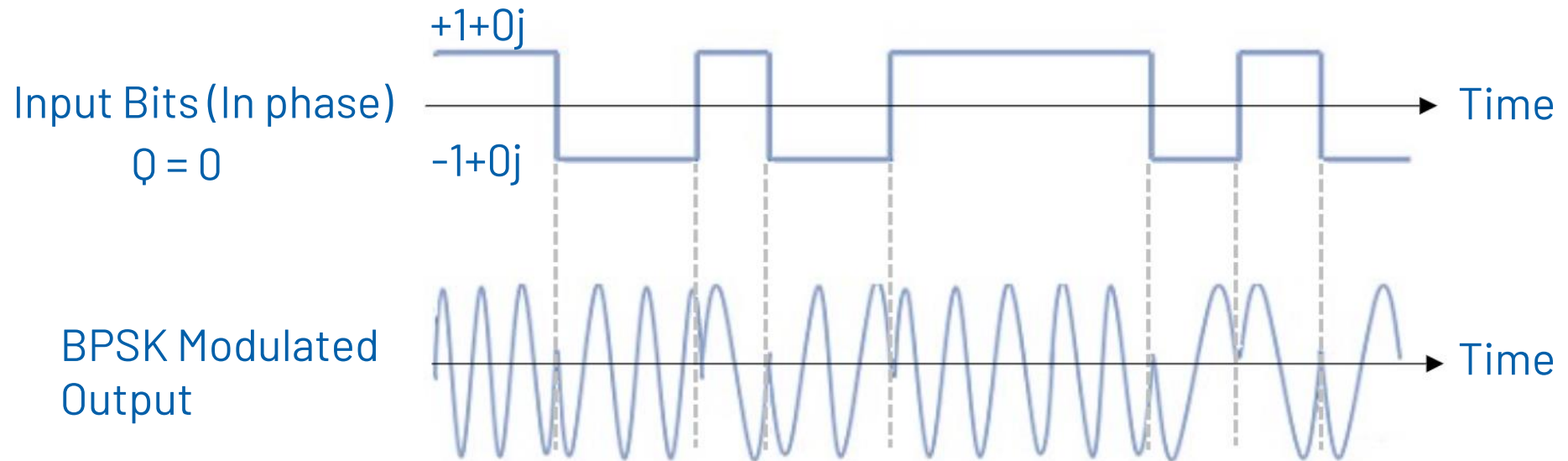
## 6. Phase Shift Keying – BPSK in Python

- Binary Shift Keying is the simplest type of Phase Modulation where the binary data (bits 0 and 1) are encoded using two distinct phase states of the carrier.

### BPSK Constellation:



### Encoding: bits $\rightarrow$ symbols \* carrier\_sinusoid



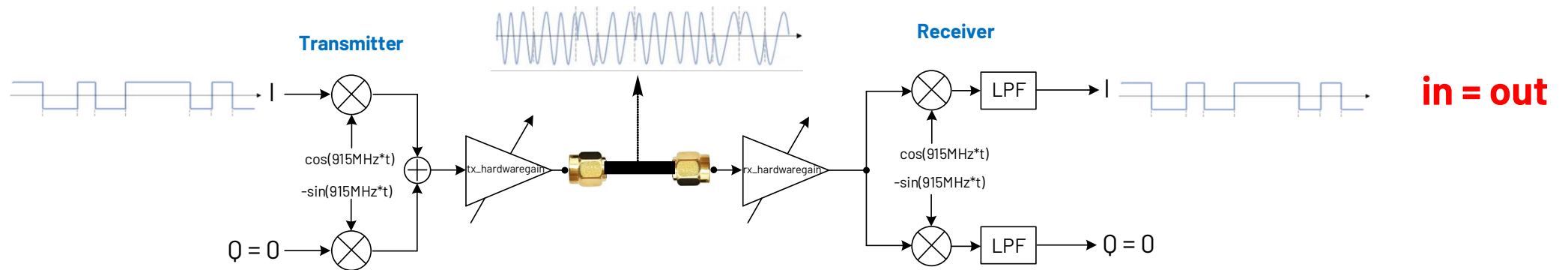
## 6. Phase Shift Keying – BPSK in Python

### ▶ **Where is Phase Modulation used?**

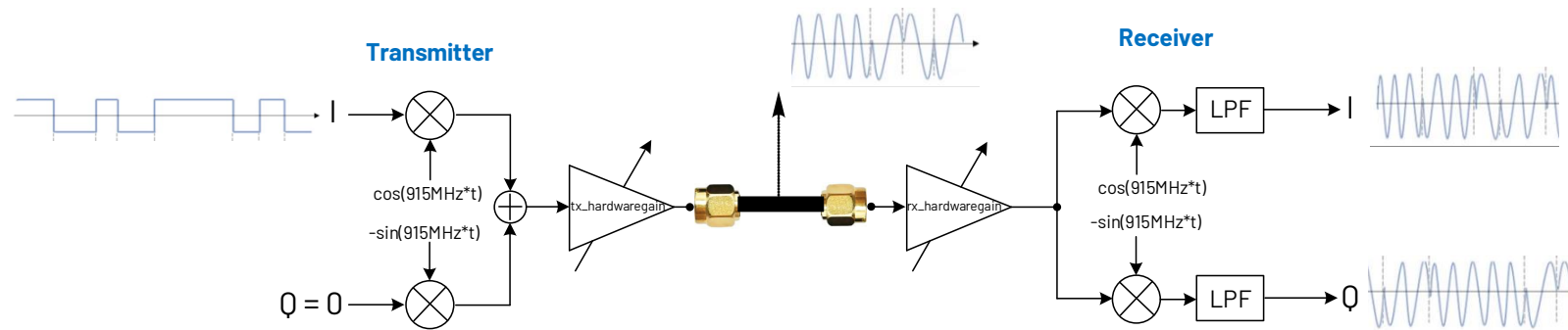
- ▶ GSM
- ▶ Satellite Television
- ▶ Wi-Fi
- ▶ Many Others

## 6. Phase Shift Keying – BPSK in Python

- In an **ideal world** where the LO's of TX and RX are perfectly in sync and where there is no time Delay between RX and TX:



- In a **real world**:



## 6. Phase Shift Keying – BPSK in Python

### ► **What are the problems when demodulating PSK signals?**

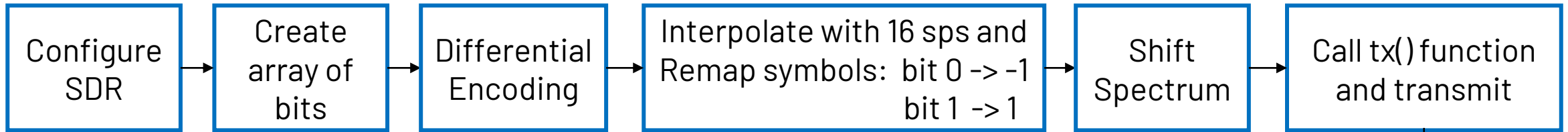
- Phase Offset of LO at RX relative to TX
- Frequency Offset of LO at RX relative to TX
- Variation of these two in time with distance change and temperature change  
(as we saw in the last example when frequency was varying with the change of distance)

► **Luckily, all these can be solved by software!**

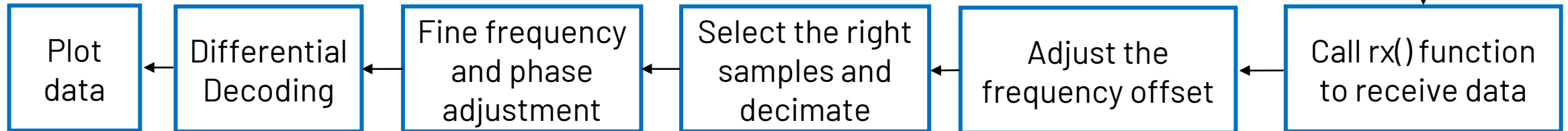
# 6. Phase Shift Keying – BPSK in Python

## ► What this code does?

### ► Transmitter:

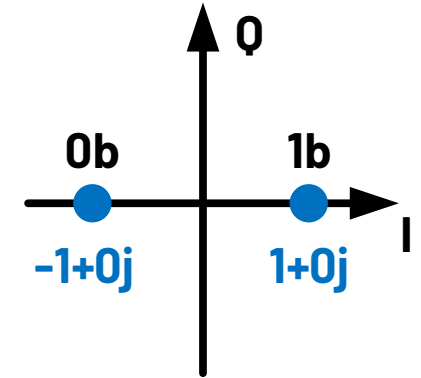


### ► Receiver:



# 6. Phase Shift Keying – BPSK in Python

- What differential encoding and decoding does?
- To transmit a bit of “1” change state (e.g. if previous is “0”, change to “1”; if previous is “1”, change to “0”).
- To transmit a bit of “0” repeat state (e.g. if previous is “1”, repeat “1”; if previous is “0”, repeat “0”).



## ► Encoder:

```

45 #####
46 # Encode array of bits with differential encoding algorithm#####
47 #####
48 # Initialize the encoded array
49 bits_encoded = np.zeros(len(bits) + 1, dtype=np.uint8)
50 # Set the first encoded bit to be the same as the first input bit
51 bits_encoded[0] = 0
52 # Differential encoding for the rest of the bits
53 for i in range(1, len(bits_encoded)):
54     bits_encoded[i] = (bits_encoded[i - 1] ^ bits[i - 1]) # XOR operation can also be used
55 #####
56 #####

```

## ► Decoder:

```

228 #####
229 # Decide if the received symbol is a 1 or a 0 #####
230 #####
231 received_bits_undecoded = np.where(samples_rx_costas_loop.real > 0, 1, 0)
232 #####
233 #####
234 #####
235 # Decode bits with differential decoding #####
236 #####
237 # Differential decoded bits
238 bits_decoded = np.zeros(len(received_bits_undecoded) - 1, dtype=np.uint8)
239 for i in range(0, len(bits_decoded)):
240     bits_decoded[i] = (received_bits_undecoded[i] ^ received_bits_undecoded[i + 1])
241 #print("Differential decoded bits:" + "\n" + str(bits_decoded))
242 #samples_rx_costas_loop = bits_decoded
243 #####
244 #####
245 #####
246 #####
247 # Extract only one packet #####
248 #####
249 packet_length = num_symbols
250 rx_packet = np.zeros(len(bits_decoded), dtype=np.uint8)
251 for i in range(len(bits_decoded) - len(predefined_key) + 1):
252     if np.array_equal(bits_decoded[i:i+len(predefined_key)], predefined_key):
253         # Extract the packet starting from this index
254         rx_packet = bits_decoded[i:i + packet_length]
255         break
256 bits_demodulated = rx_packet
257 #####
258 #####

```



## 6. Phase Shift Keying – BPSK in Python

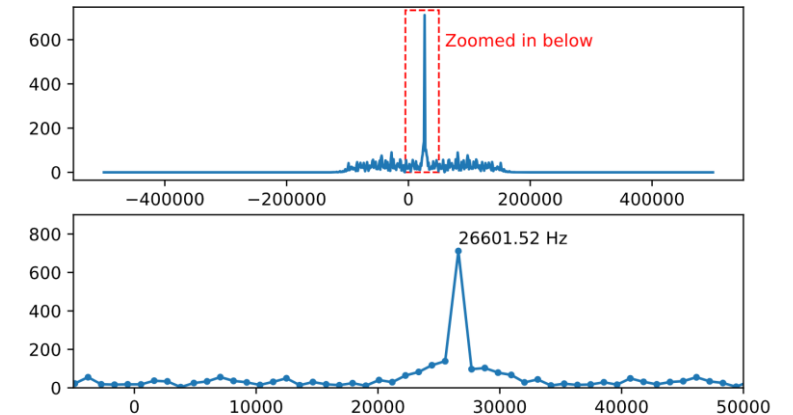
- How we adjust the frequency offset?
- First square the received signal => all symbols  $(s(t))^2$  will have a constant positive value

$$r^2(t) = s^2(t)e^{j4\pi f_o t}$$

```
136 # Coarse frequency adjustment
137 samples_adjust = samples**2 # square the received signal to obtain 2* offset frequency
```

- Take the FFT and measure the peak => the measured peak will be at 2\*offset\_frequency

```
133 #####
134 # Adjust frequency offset #####
135 #####
136 # Coarse frequency adjustment
137 samples_adjust = samples**2 # square the received signal to obtain 2* offset frequency
138 psd = np.fft.fftshift(np.abs(np.fft.fft(samples_adjust)))
139 f = np.linspace(-fs/2.0, fs/2.0, len(psd))
140 max_freq = f[np.argmax(psd)]
141 Ts = 1/fs # calc sample period
142 t = np.arange(0, Ts*len(samples), Ts) # create time vector
143 samples = samples * np.exp(-1j*2*np.pi*max_freq*t/2.0)
```



- Apply frequency correction on received samples based on the above measurement

```
141 Ts = 1/fs # calc sample period
142 t = np.arange(0, Ts*len(samples), Ts) # create time vector
143 samples = samples * np.exp(-1j*2*np.pi*max_freq*t/2.0)
```

`max_freq` is the frequency where the peak power was measured

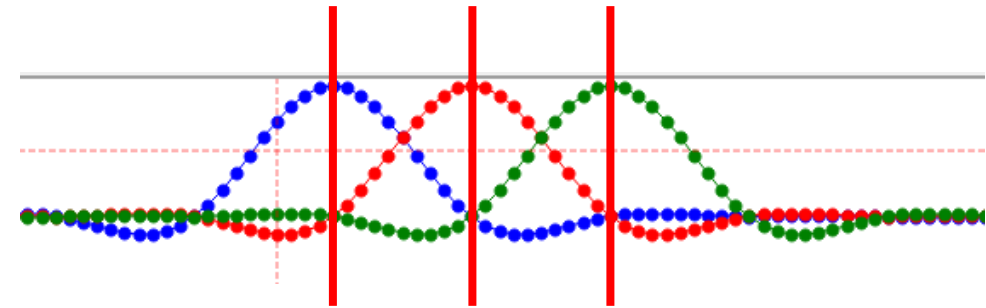
## 6. Phase Shift Keying – BPSK in Python

- How do we select the right samples (Mueller and Muller timing recovery technique [6, 10])?
- In the below code, the variable “mu” represents the timing offset we have to add to 16sps because we have to select one from each 16 samples. For example, if  $\mu = 2.43 \Rightarrow$  we have to shift the input by 2.43 samples.
- After a few iterations of the while loop, “mu” stabilizes and only the correct samples should be pulled.

```

162 #####
163 # Select only the right samples and decimate #####
164 #####
165 samples_interpolated = signal.resample_poly(samples, 16, 1) # interpolation
166 mu = 0 # initial estimate of phase of sample
167 out = np.zeros(len(samples) + 10, dtype=complex)
168 out_rail = np.zeros(len(samples) + 10, dtype=complex) # stores values, each iteration we need the previous
169 # 2 values plus current value
170 i_in = 0 # input samples index
171 i_out = 2 # output index (let first two outputs be 0)
172 while i_out < len(samples) and i_in+16 < len(samples):
173     out[i_out] = samples[i_in + int(mu)] # grab what we think is the "best" sample
174     out[i_out] = samples_interpolated[i_in*16 + int(mu*16)]
175     out_rail[i_out] = int(np.real(out[i_out]) > 0) + 1j*int(np.imag(out[i_out]) > 0)
176     x = (out_rail[i_out] - out_rail[i_out-2]) * np.conj(out[i_out-1])
177     y = (out[i_out] - out[i_out-2]) * np.conj(out_rail[i_out-1])
178     mm_val = np.real(y - x)
179     mu += sps + 0.3*mm_val
180     i_in += int(np.floor(mu)) # round down to nearest int since we are using it as an index
181     mu = mu - np.floor(mu) # remove the integer part of mu
182     i_out += 1 # increment output index
183 out = out[2:i_out] # remove the first two, and anything after i_out (that was never filled out)
184 samples = out # only include this line if you want to connect this code snippet with the Costas Loop later on
185 # Save samples after time adjustment with interpolation for plots
186 samples_rx_time_adj = samples
187 #####
188 #####

```



We want to sample where the adjacent symbols cross 0 and discard in between samples.

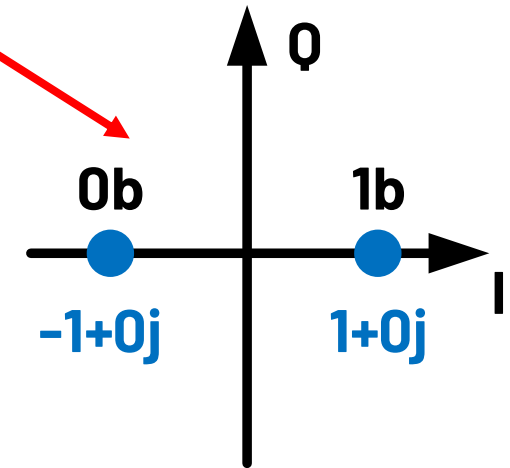
## 6. Phase Shift Keying – BPSK in Python

- How Fine Frequency Synchronization is done (Costas Loop [6], [11])?
- It functions like a PLL. We multiply the real part of the sample (I) by the imaginary part (Q), and because Q should be equal to zero for BPSK, the error function is minimized when there is no phase or frequency offset that causes energy to shift from I to Q (**Q is the error signal, adjust and keep Q = 0**).

```

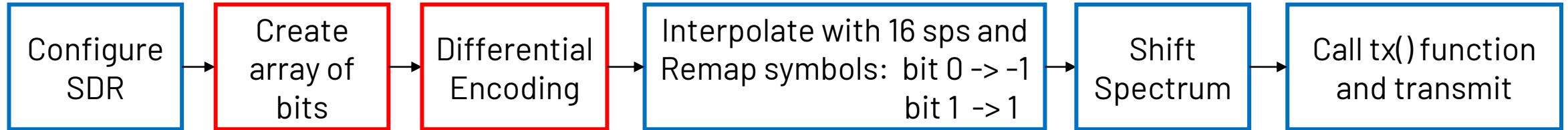
190 #####
191 # Fine frequency and phase adjustment (Costas Loop) #####
192 #####
193 N = len(samples)
194 phase = 0
195 freq = 0
196 # These next two params is what to adjust, to make the feedback loop faster or slower (which impacts stability)
197 alpha = 0.332
198 beta = 0.01932
199
200 out = np.zeros(N, dtype=complex)
201 freq_log = []
202
203 for repeat in range(1):
204     for i in range(N):
205         out[i] = samples[i] * np.exp(-1j*phase) # adjust the input sample by the inverse of the estimated phase offset
206         error = np.real(out[i]) * np.imag(out[i]) # This is the error formula for 2nd order Costas Loop (e.g. for BPSK)
207
208         # Debugging output
209         # print(f"Iteration {i}: Phase={phase}, Freq={freq}, Error={error}")
210
211         # Advance the loop (recalc phase and freq offset)
212         freq += (beta * error)
213         freq_log.append(freq * fs / (2*np.pi)) # convert from angular velocity to Hz for logging
214         phase += freq * (alpha * error)
215
216         # Optional: Adjust phase so its always between 0 and 2pi, recall that phase wraps around every 2pi
217         while phase >= 2*np.pi:
218             phase -= 2*np.pi
219         while phase < 0:
220             phase += 2*np.pi
221
222 samples = out
223
224 # Save the samples after costas loop (fine frequency and phase adjustment) for plots
225 samples_rx_costas_loop = samples
226 #####

```

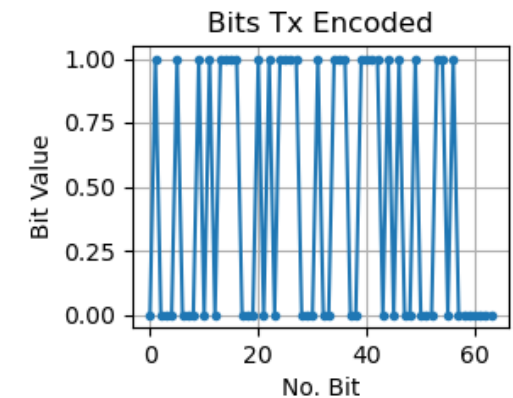
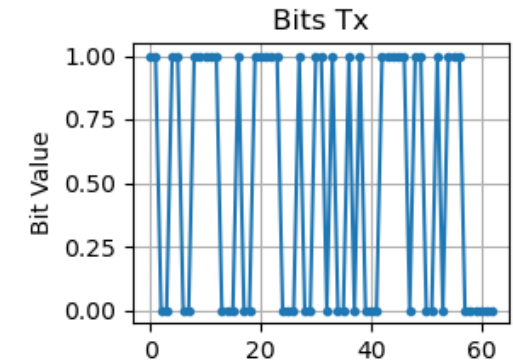


# 6. Phase Shift Keying – BPSK in Python

## ► Transmitter:

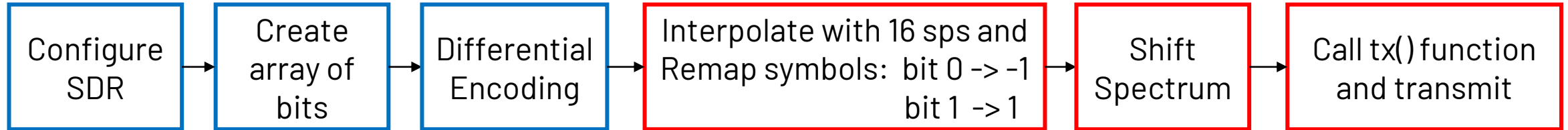


► **Create the array of bits and encode them with Differential Encoding (this is the data you want to transmit):**

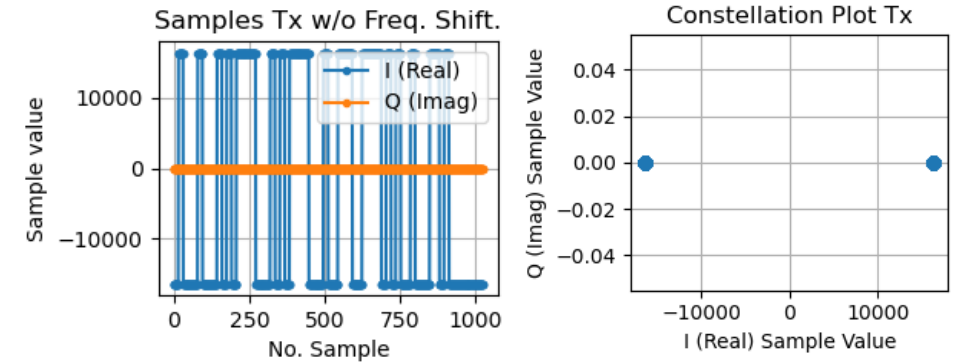


# 6. Phase Shift Keying – BPSK in Python

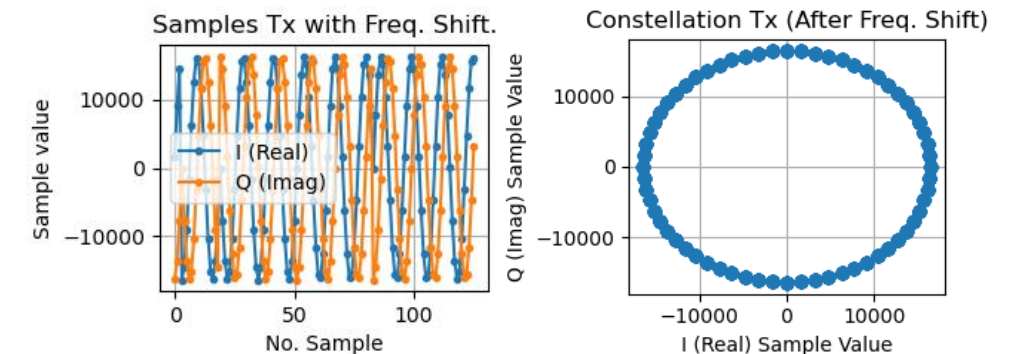
## ► Transmitter:



- Repeat each bit 16 times (interpolate) => 16sps
- For BPSK the data is complex ( $Q = 0$ )
- This is ideally what we want to obtain at receiver ( $Q = 0$ , constellation symbols:  $-1+0j$ ;  $1+0j$ ).

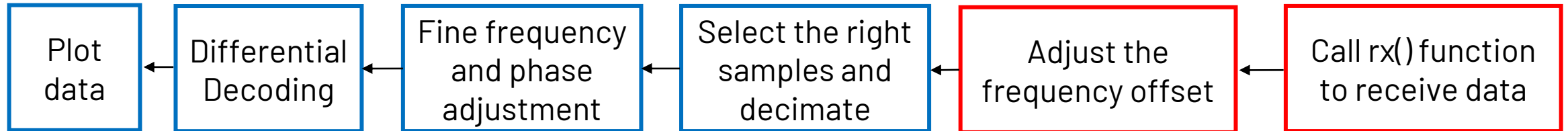


- Shift spectrum to a higher frequency by multiplying with a complex sinusoid.
- Call `sdr.tx()` function to transmit the data

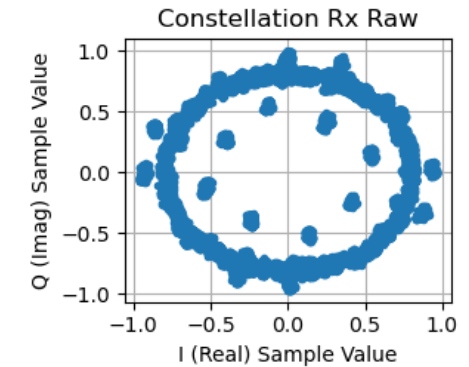
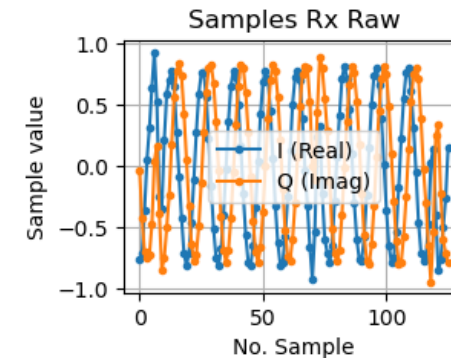


# 6. Phase Shift Keying – BPSK in Python

## ► Receiver



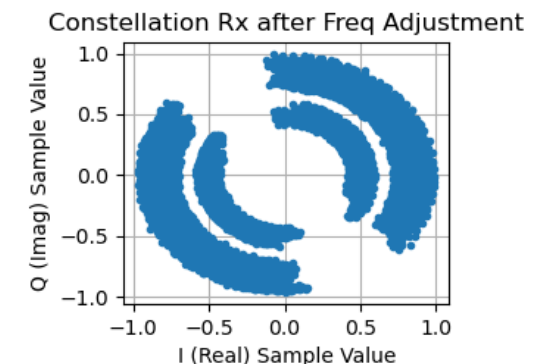
- This is what we get after receive.
- Observe the phase offset and frequency offset that needs to be corrected,  $Q \neq 0$ .



- After we adjust the frequency offset, we don't see so much of the frequency difference in the received signal, but the phase of the I and Q signals are still not right.

- Observe samples that fall in the middle due to inaccurate sampling.

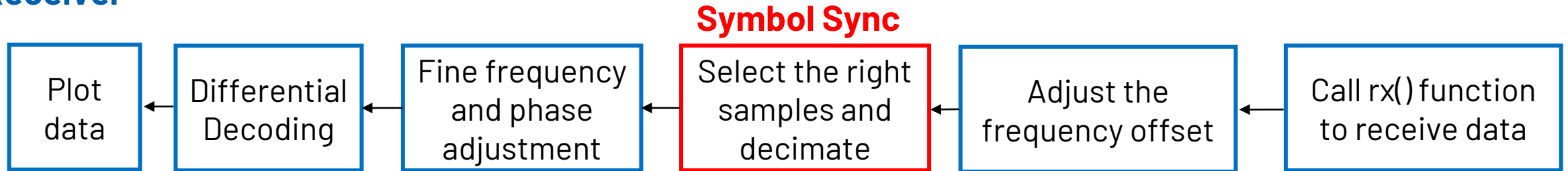
- $Q \neq 0$





# 6. Phase Shift Keying – BPSK in Python

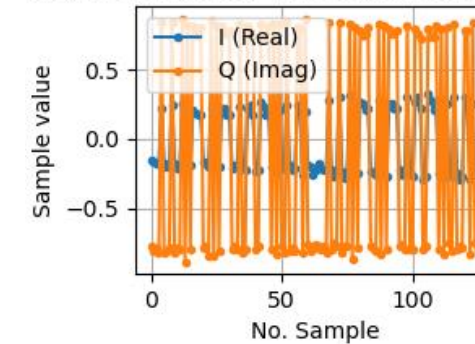
## ► Receiver



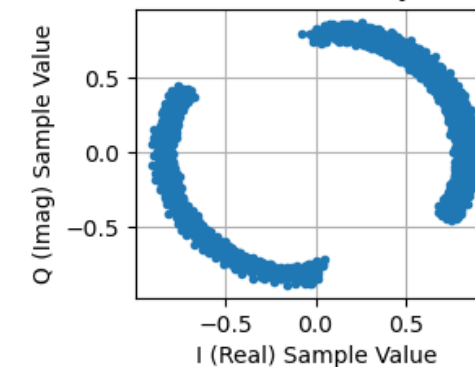
► After selecting the right samples, we get rid of the ones that fell in the middle of the constellation but a small phase and frequency offset is still present.

►  $Q \neq 0$

Samples Rx after Freq Adj. & Symbol Sync

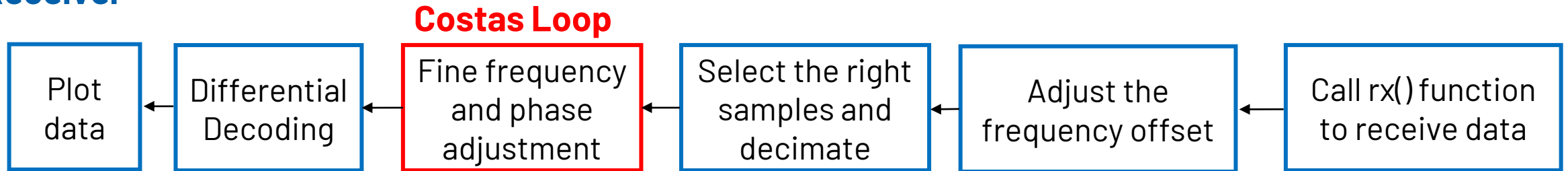


Constellation Rx after Symbol Sync



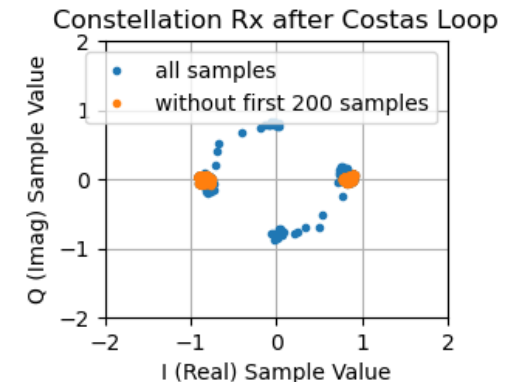
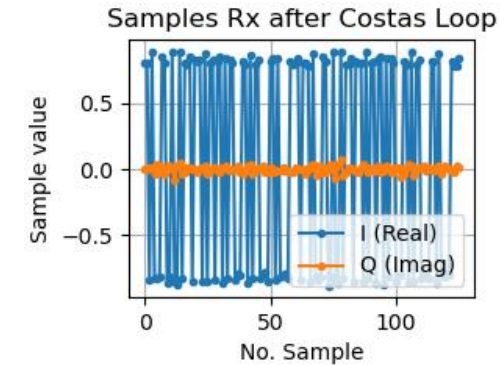
# 6. Phase Shift Keying – BPSK in Python

## ► Receiver



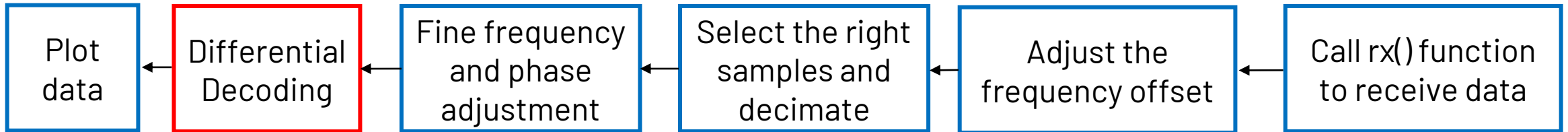
► **Observe that after fine tuning the frequency and the phase, we get the right samples with a few errors that appear before the Costas Loop locks onto the right frequency and phase.**

► **Now  $Q = 0$ .**

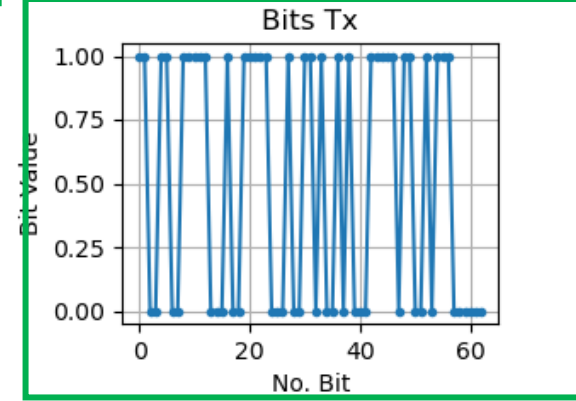
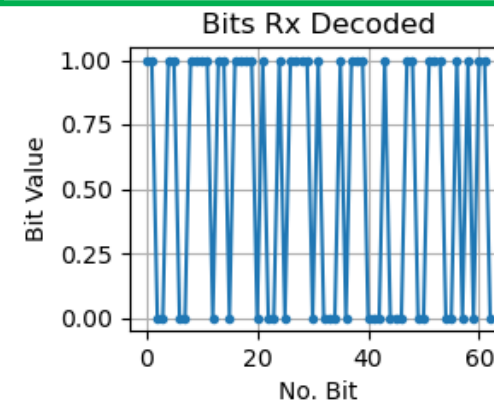
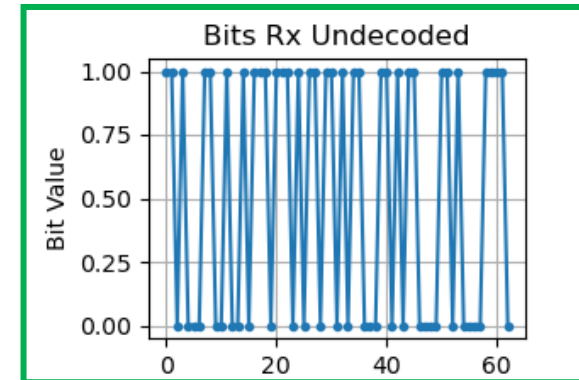


# 6. Phase Shift Keying – BPSK in Python

## ► Receiver



► After decoding with differential decoding, observe that “Bits Rx Decoded” are the same as “Bits Tx”.

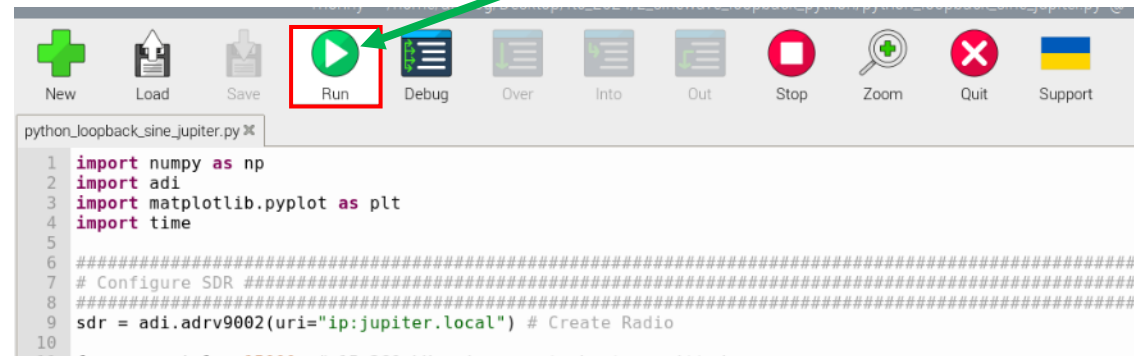


## 6. Phase Shift Keying – BPSK in Python

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.

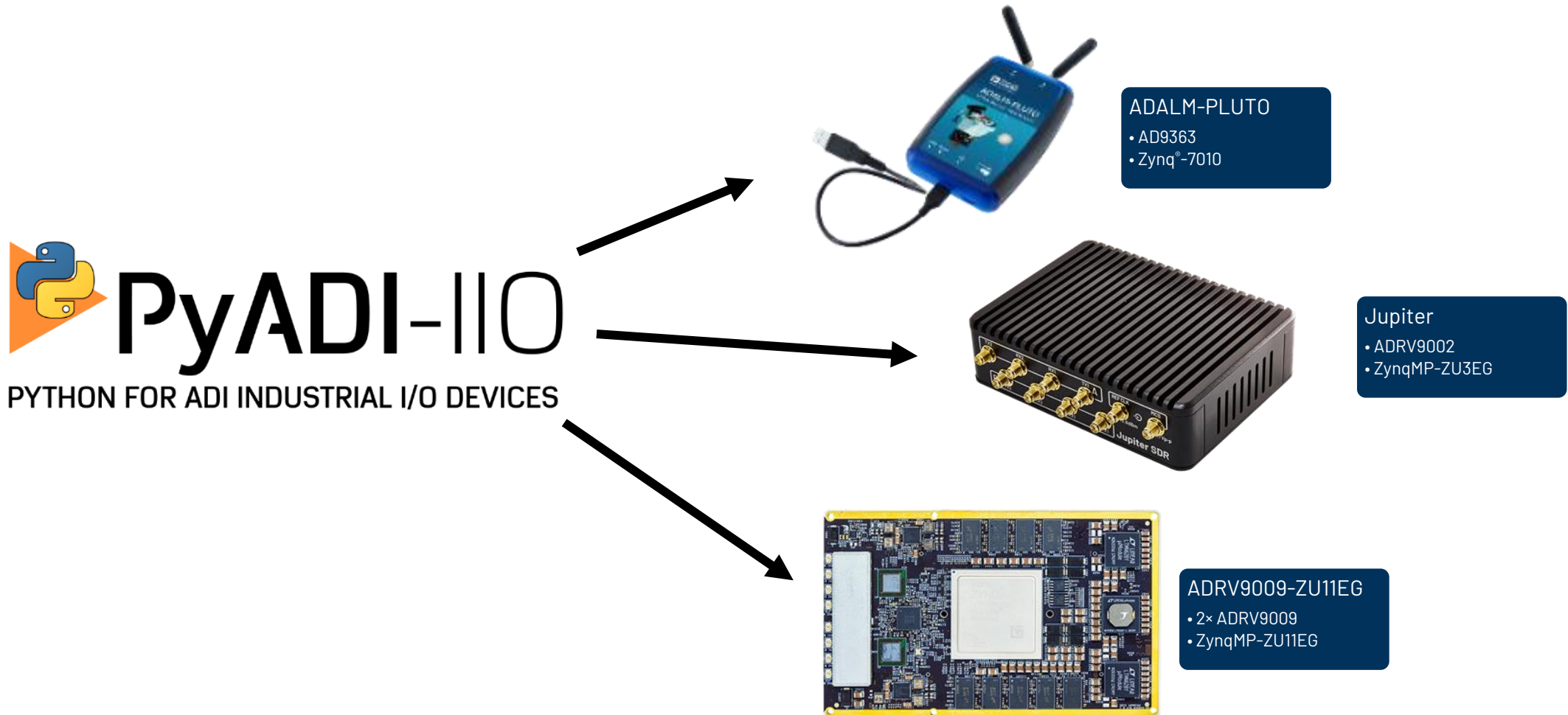


- Open “bpsk\_loopback\_jupiter.py” file using “Thonny” IDE from:  
“/home/analog/Desktop/ftc\_2024/4\_bpsk\_loopback\_python/”
- To run the python script, click on the **“Run”** button:



## 6. Phase Shift Keying – BPSK in Python

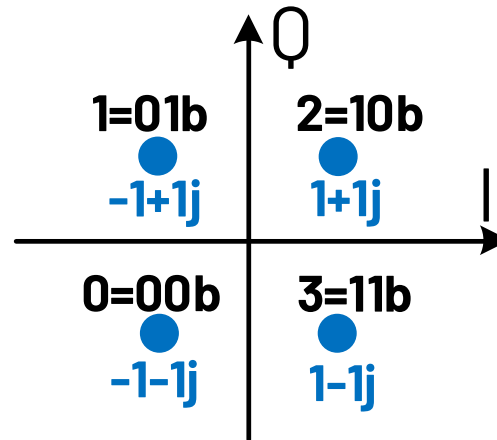
- Using PyADI-II0 you can reuse the code and run it on different SDRs like Pluto or Talise.



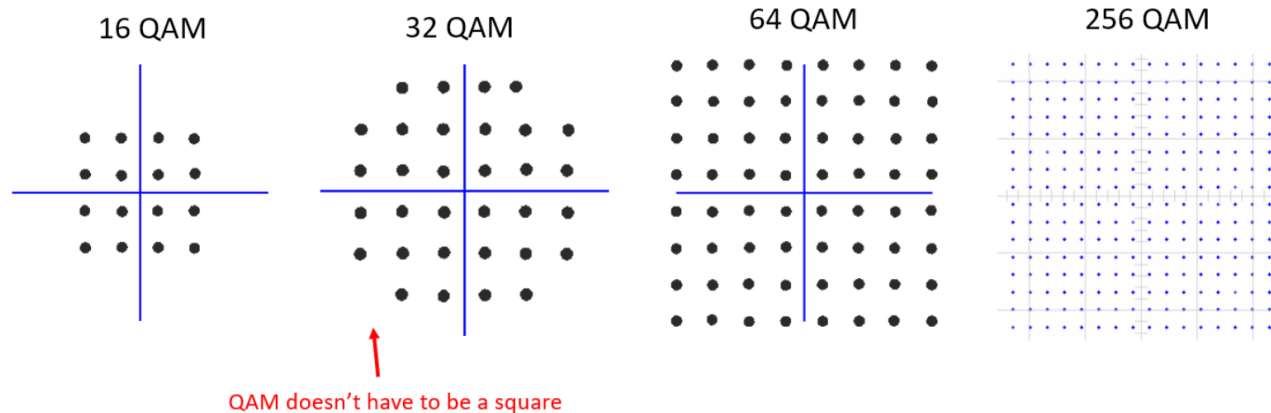
# 7. Phase Shift Keying – QPSK in GNU Radio

# 7. Phase Shift Keying – QPSK in GNU Radio

- QPSK is similar with BPSK but has more symbols.



- PSK with more than 4 symbols (here the amplitude is modulated too):







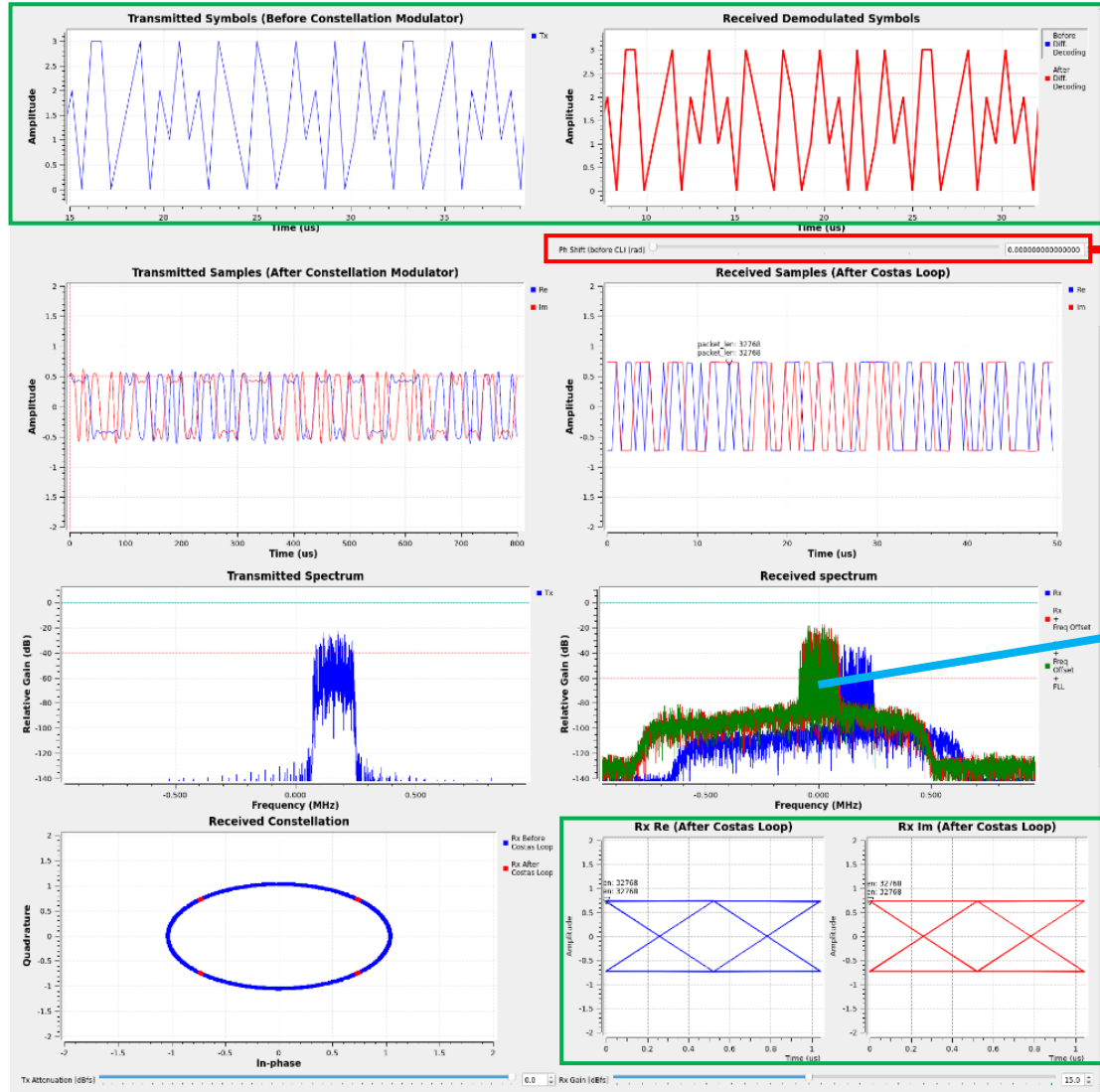
## Symbol Sync

## Differential Decoder

## Frequency Locked Loop

→ **Constellation Receiver (Costas Loop)**

# 7. Phase Shift Keying – QPSK in GNU Radio



Same pattern of symbols  
at TX and demodulated RX

From this slider you can rotate the  
constellation in steps of 90 deg => observe  
that the symbols received are the same due  
to Differential Decoding

With green, you can see the  
RX Spectrum after FLL block

Here is the Eye diagram after  
Costas loop of the Rx

## 7. Phase Shift Keying – QPSK in GNU Radio

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

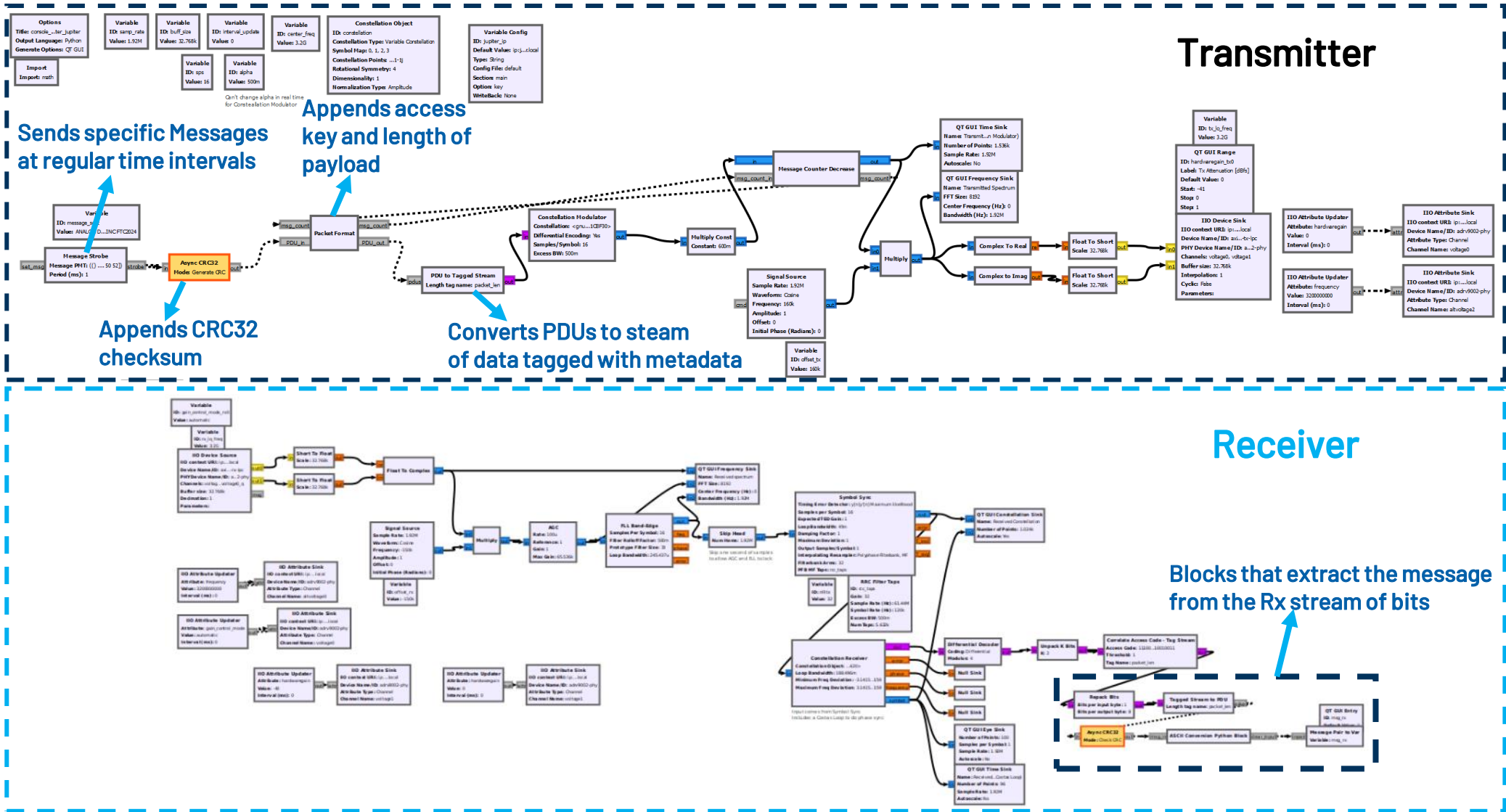
```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

- In GNU Radio companion app, open from File -> Open:  
`/home/analog/Desktop/ftc_2024/5_qpsk_loopback_gnuradio/qpsk_loopback_jupiter.grc`

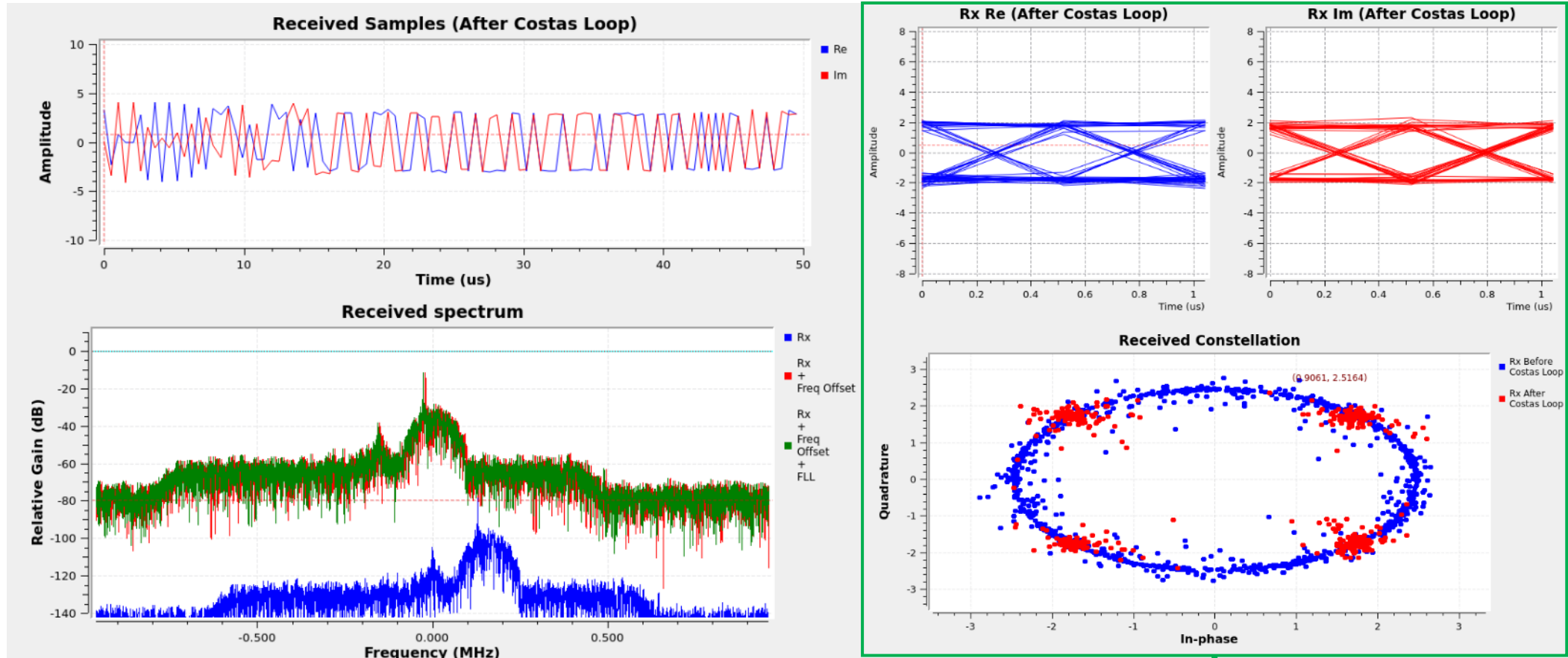
# 8. Phase Shift Keying – Receive a message in GNU Radio

# 8. Phase Shift Keying – Receive a message in GNU Radio



# 8. Phase Shift Keying – Receive a message in GNU Radio

'msg\_rx': 'ANALOG DEVICES INC FTC2024'



The Symbols are Noisier because the data is sent in bursts (not continuously)



## 8. Phase Shift Keying – Receive a message in GNU Radio

- For this example, connect **only one antenna** to the RX1 of Jupiter



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

**We will transmit to you**

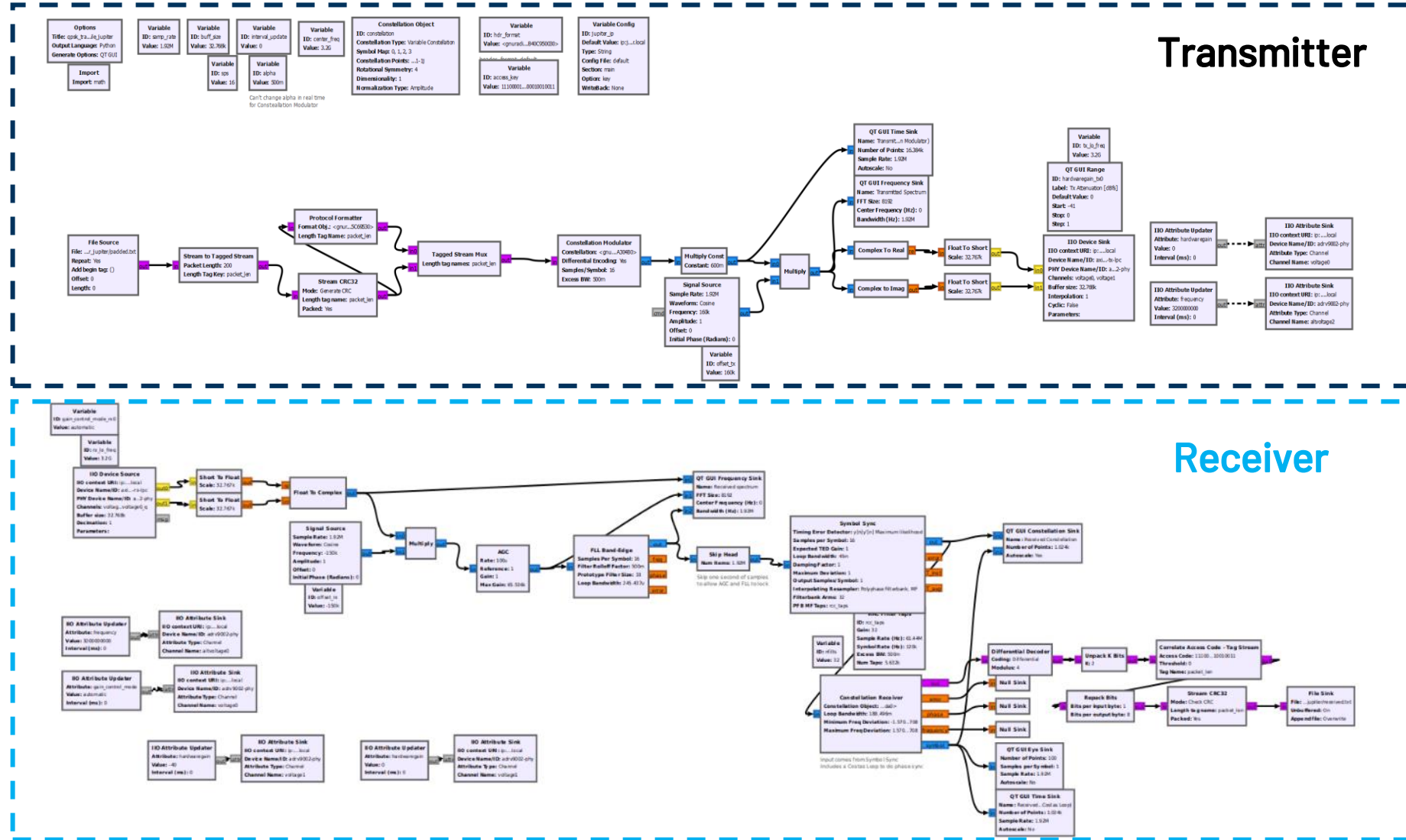
- In GNU Radio companion app, open from File -> Open:

`/home/analog/Desktop/ftc_2024/6_console_message_point_to_point_gnuradio/receiver_jupiter/console_message_receiver_jupiter.grc`

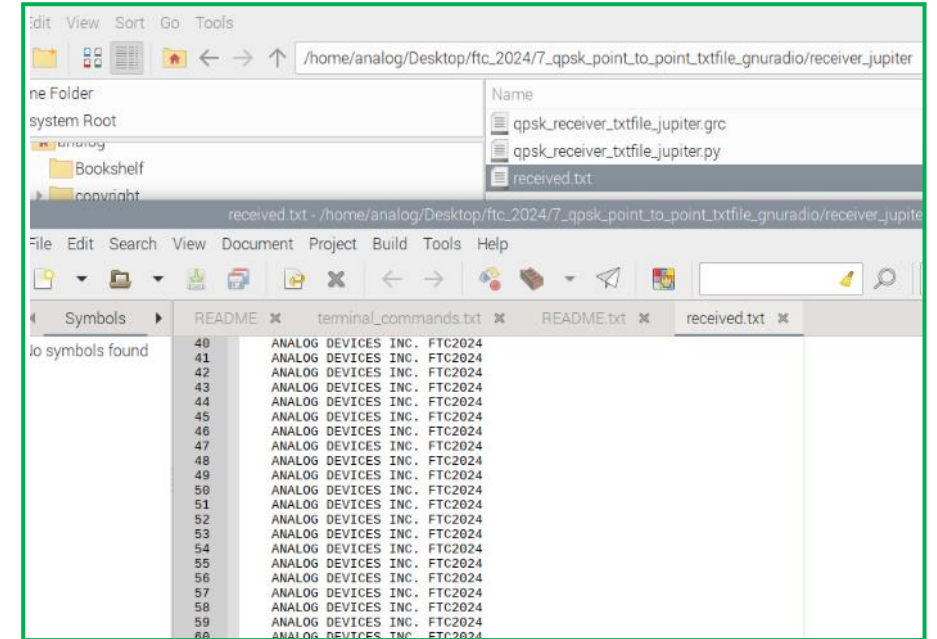
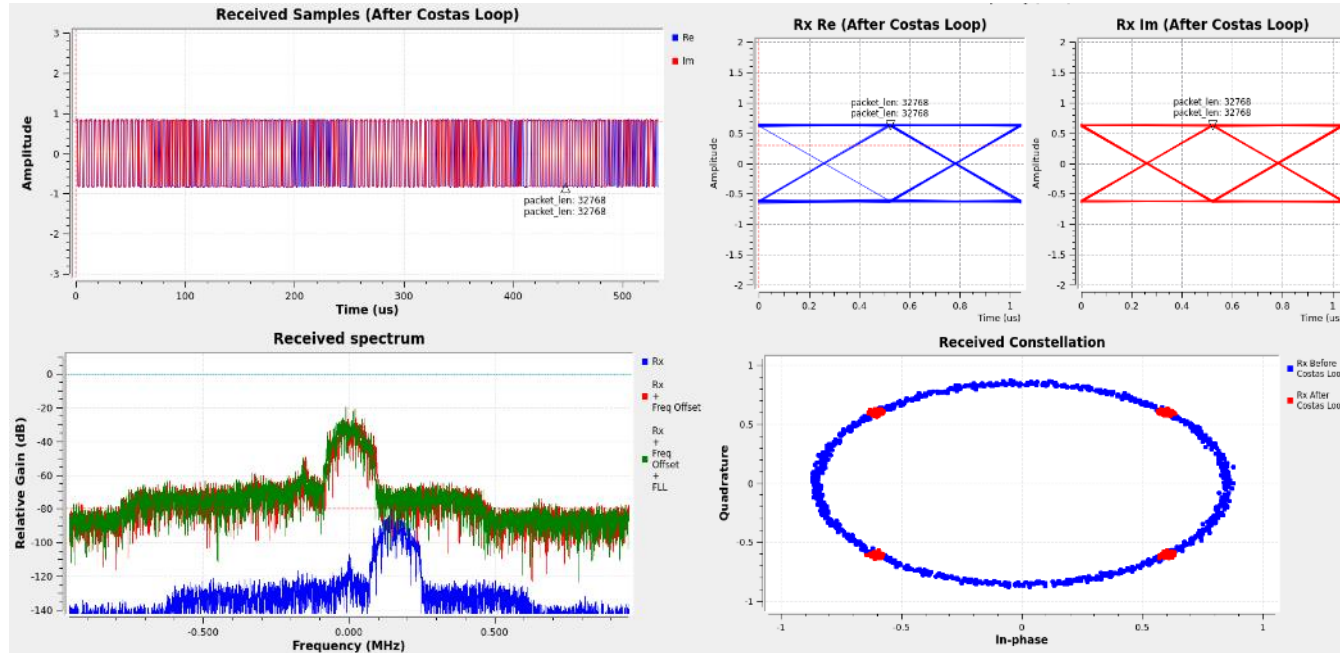


# 9. Phase Shift Keying – Receive a message and store it in a file

# 9. Phase Shift Keying – Receive a message and store it in a file



## 9. Phase Shift Keying – Receive a message and store it in a file



After you run the flowgraph for a few seconds, stop it and open the **receive.txt** file from:

**/home/analog/Desktop/ftc\_2024/7\_qpsk\_point\_to\_point\_txtfile\_gnuradio/receiver\_jupiter**

## 9. Phase Shift Keying – Receive a message and store it in a file

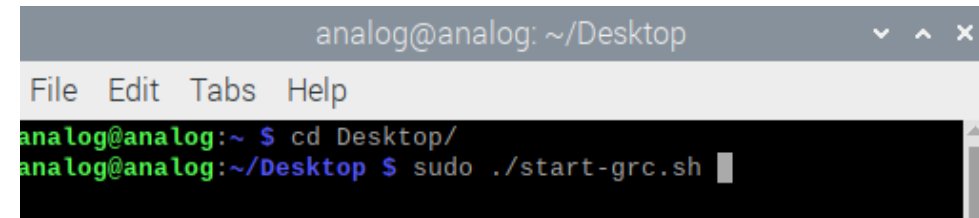
- For this example, connect **only one antenna** to the RX1 of Jupiter



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

```
$ sudo ./start-grc.sh
```



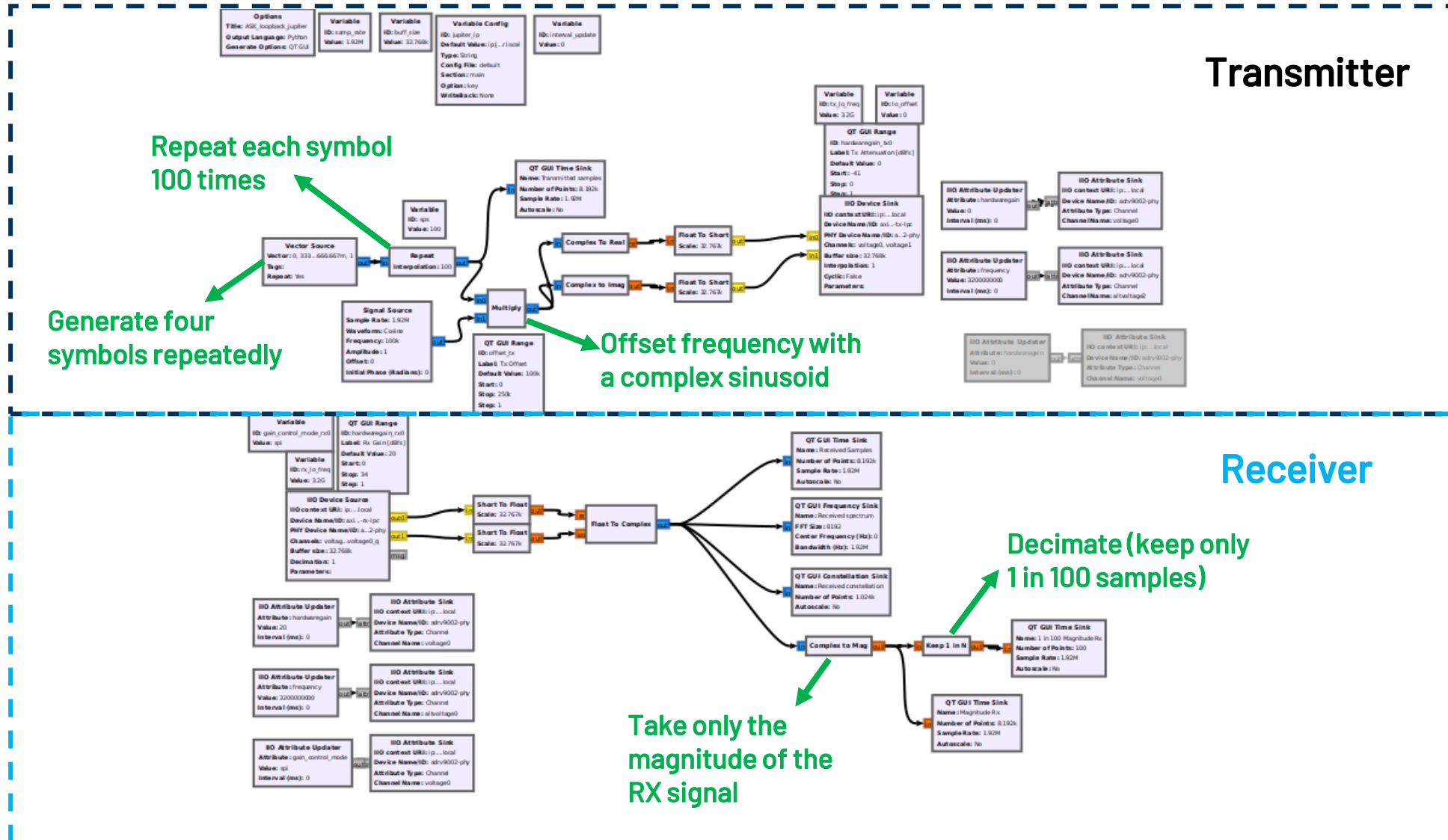
**We will transmit to you**

- In GNU Radio companion app, open from File -> Open:

```
/home/analog/Desktop/ftc_2024/7_qpsk_point_to_point_txtfile_gnuradio/receiver_jupiter/qpsk_receiver_txtfile_jupiter.grc
```

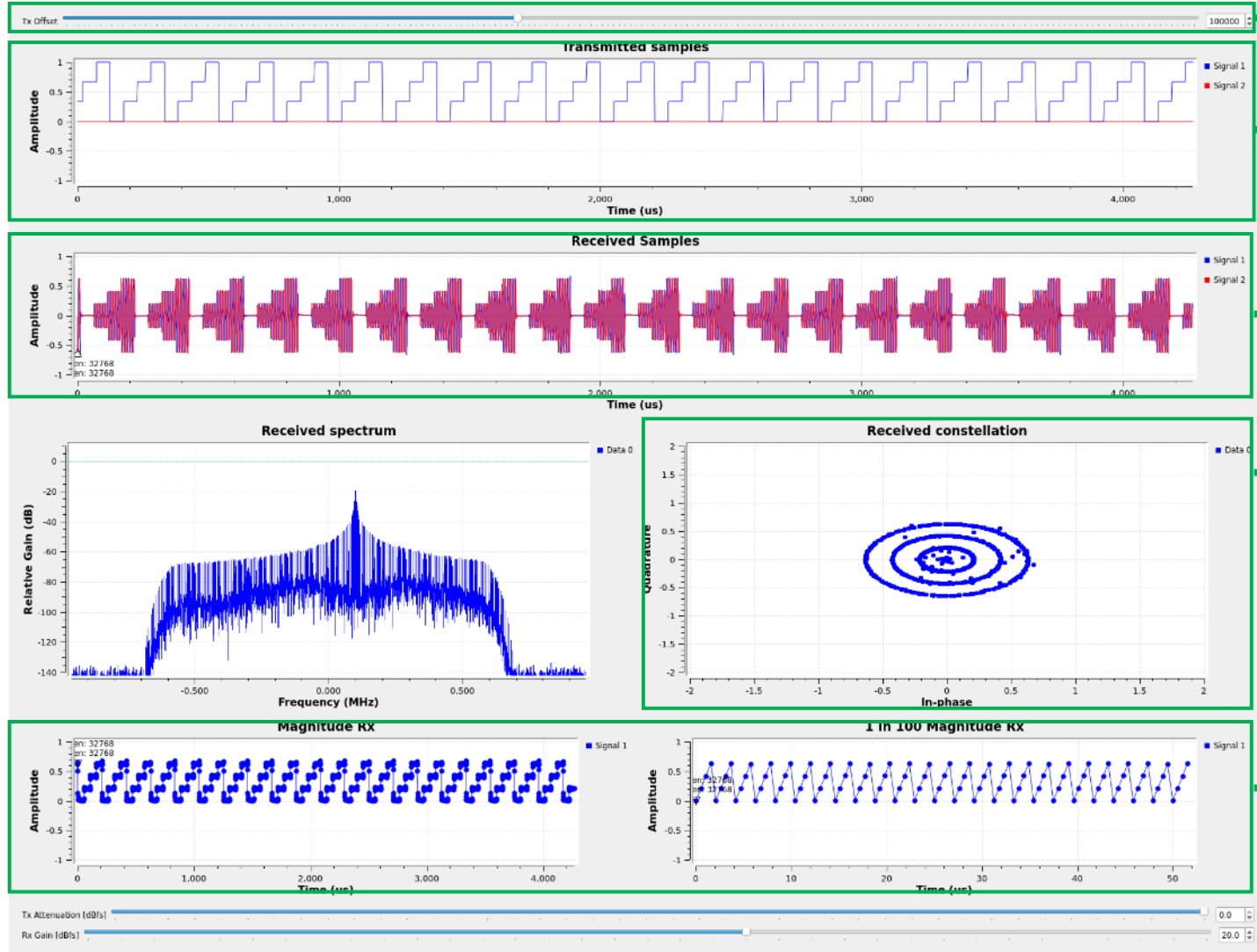
# 10. Amplitude Shift Keying – GNU Radio Example

# 10. Amplitude Shift Keying – GNU Radio Example





# 10. Amplitude Shift Keying – GNU Radio Example



Using this slider, you can offset the Tx signal relative to TX LO frequency

TX symbols: 100 sps for each symbol between (0, 1/3, 2/3, 1)

RX unprocessed signal

Observe on the RX constellation, 4 levels of amplitude

Only the magnitude of the RX signal (on right -> decimated)



## 10. Amplitude Shift Keying – GNU Radio Example

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

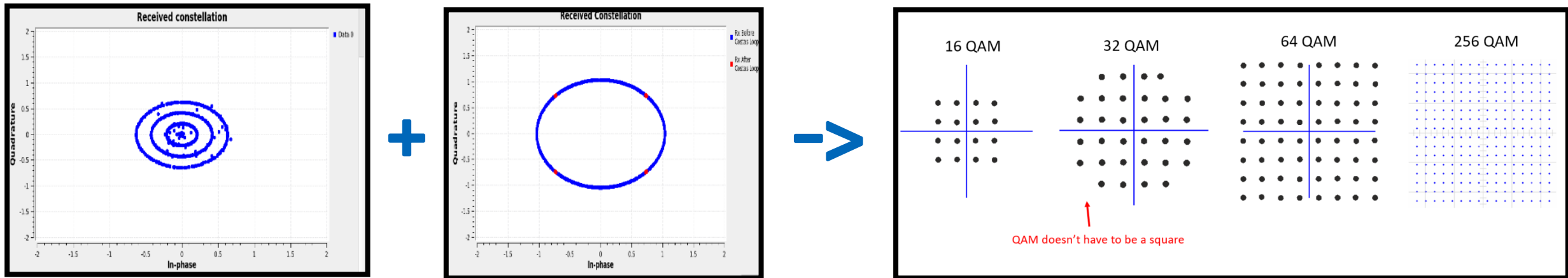
- In GNU Radio companion app, open from File -> Open:

**/home/analog/Desktop/ftc\_2024/8\_ask\_loopback\_gnuradio/ASK\_loopback\_jupiter.grc**

# 10. Amplitude Shift Keying – GNU Radio Example

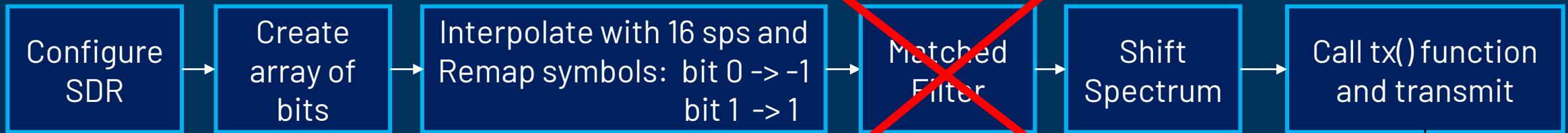
## ► Conclusion:

ASK combined with PSK forms QAM:

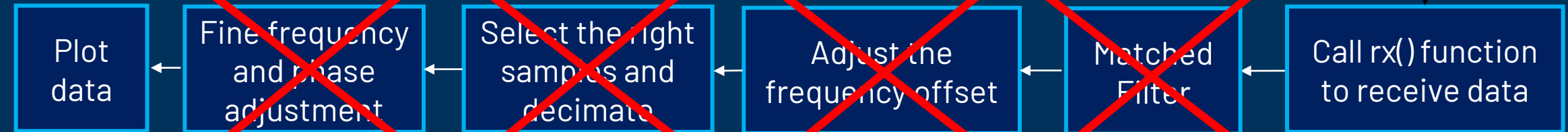


# 11. QPSK without additional digital processing – GNU Radio

## ► Transmitter:

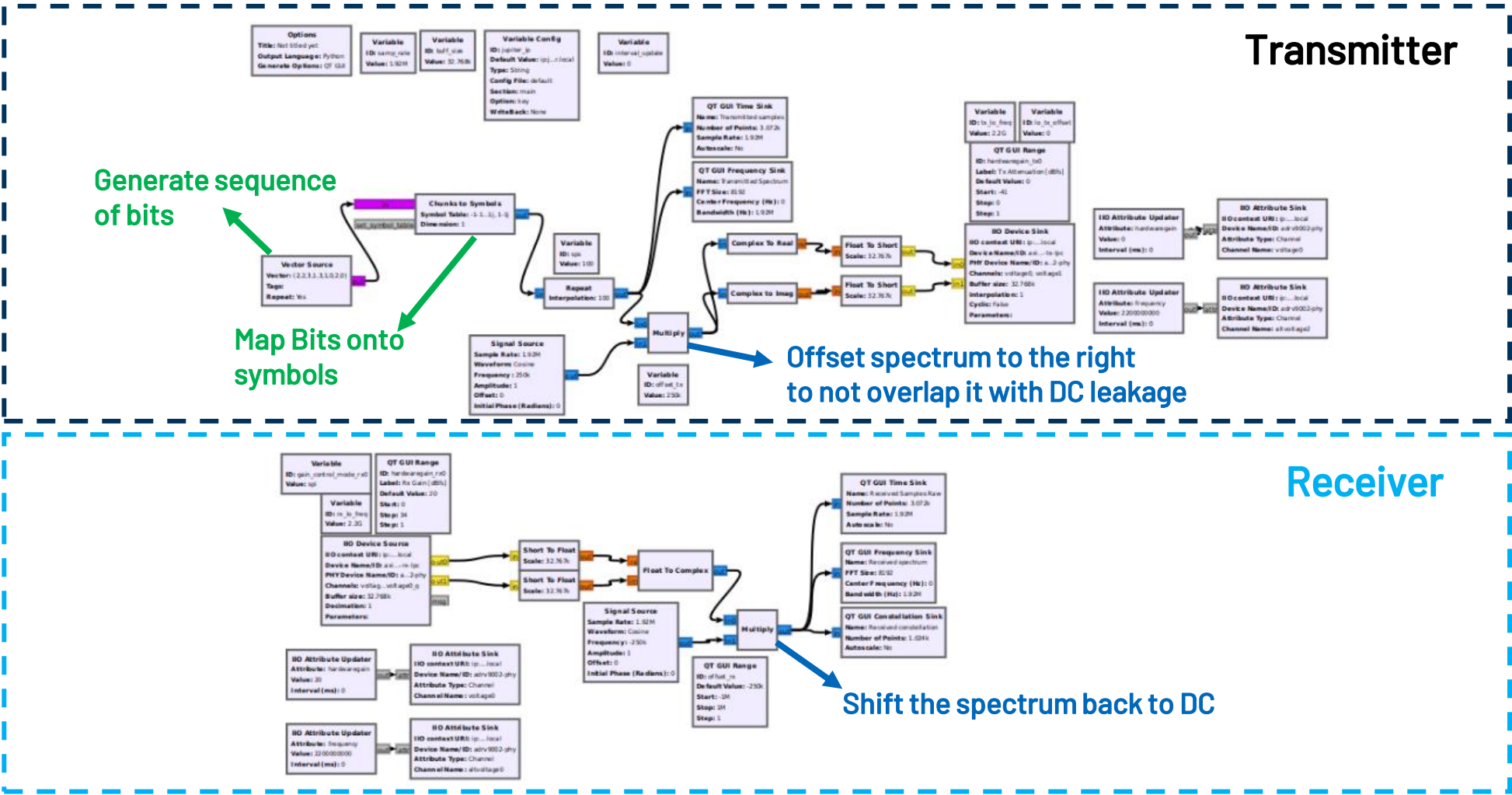


## ► Receiver:

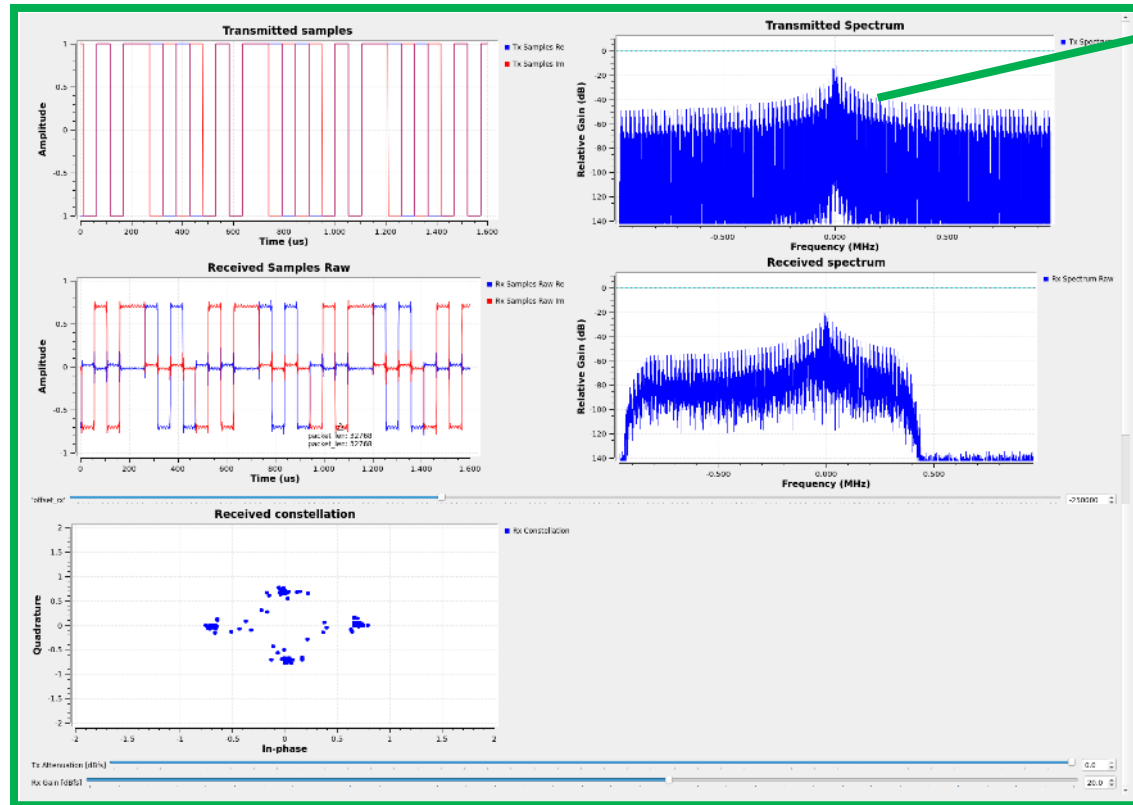


**MISSING**

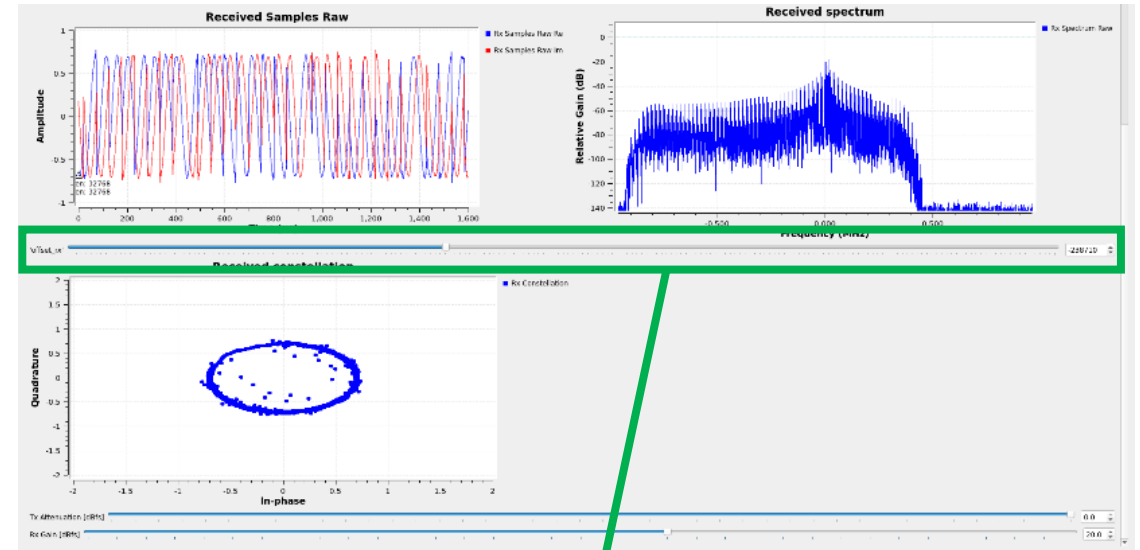
# 11. QPSK without additional digital processing – GNU Radio



# 11. QPSK without additional digital processing – GNU Radio



The spectrum is inefficiently used (spectrum of square wave)



Try tweaking the frequency offset between TX and RX and see how the data looks -> this happens when we transmit and receive between two different devices.

No frequency offset because the example works on loopback and the LO is the same for RX and TX but has a phase offset between the two paths.



# 11. QPSK without additional digital processing – GNU Radio

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

- In GNU Radio companion app, open from File -> Open:

**/home/analog/Desktop/ftc\_2024/10\_qpsk\_raw\_loopback\_gnuradio/QPSK\_raw\_loopback\_jupiter.grc**

# 11. QPSK without additional digital processing – GNU Radio

## ► Conclusions:

- The frequency offset and LO drift is required to be corrected at receiver
- The phase offset of the LO at the receiver is required to be corrected
- These are varying with distance and temperature in time => need some sort of feedback loop to constantly adjust the frequency and phase
- The correct sample from the received signal needs to be extracted (not transitions)
- Even if we do all these, if we rotate the constellation 180 deg, the symbols received will be out of place

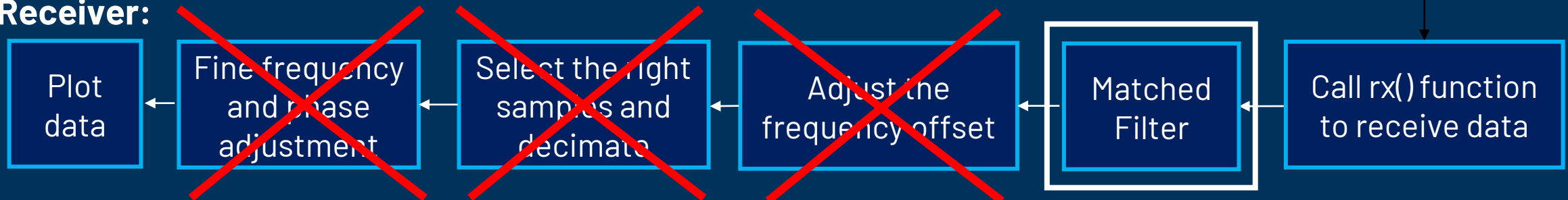


# 12. QPSK – Constellation Modulator in GNU Radio

## ► Transmitter:



## ► Receiver:



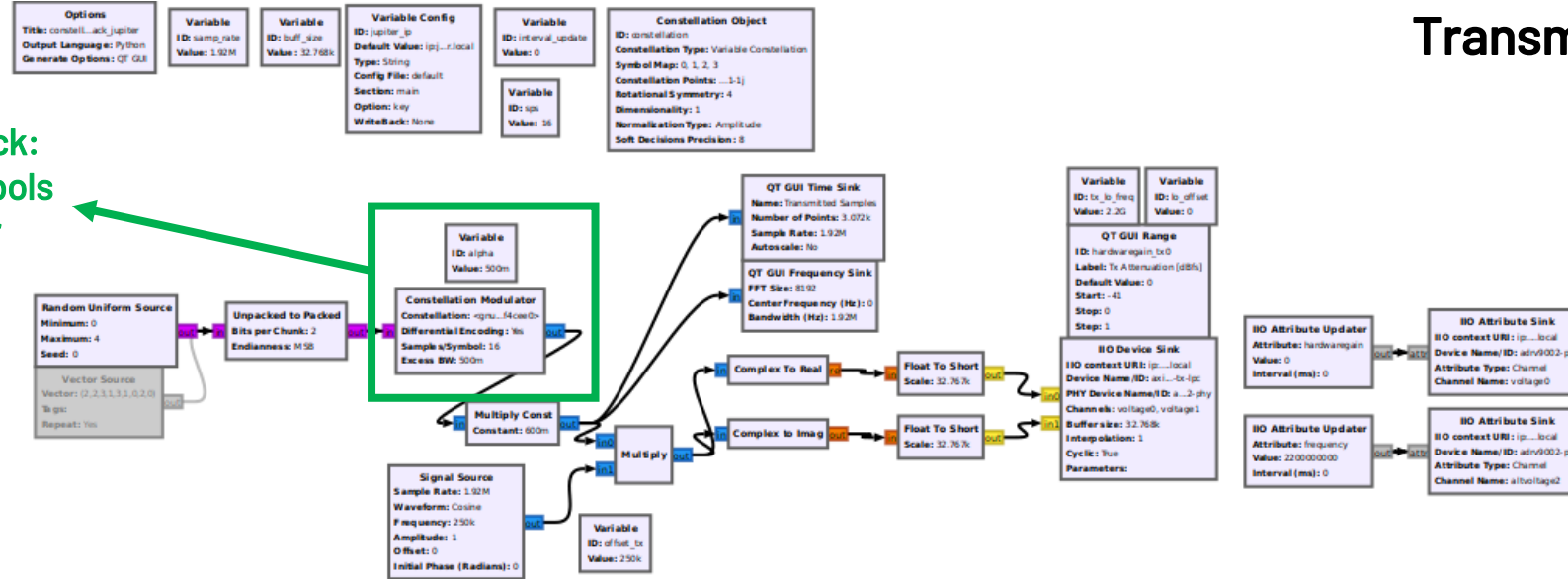
**MISSING**

**ADDED**

# 12. QPSK - Constellation Modulator in GNU Radio

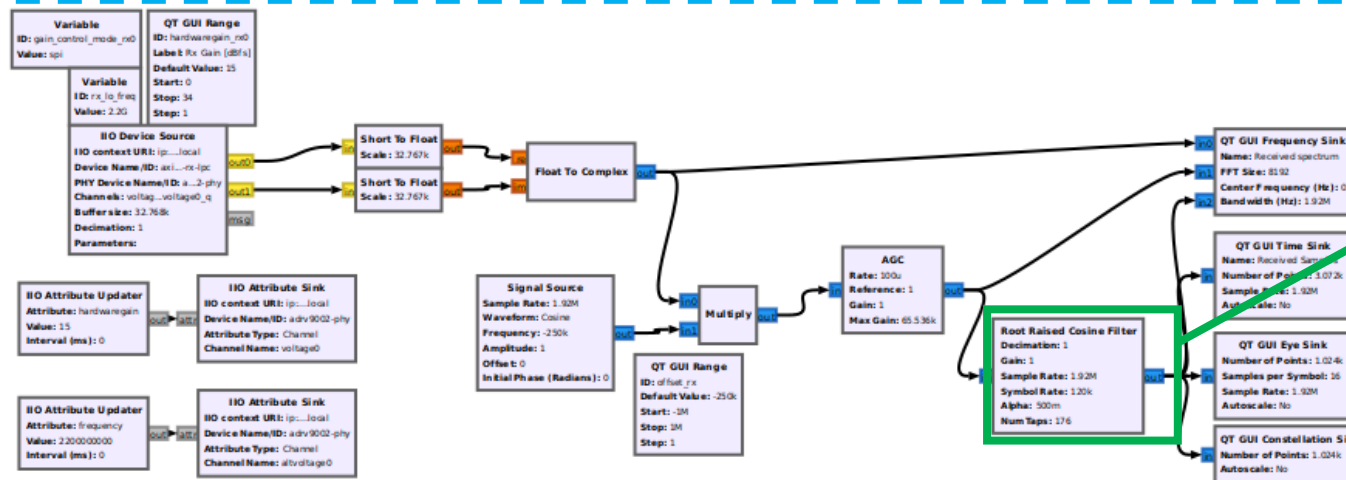
## Transmitter

Const. Modulator block:  
maps bytes into symbols  
and applies RRC filter

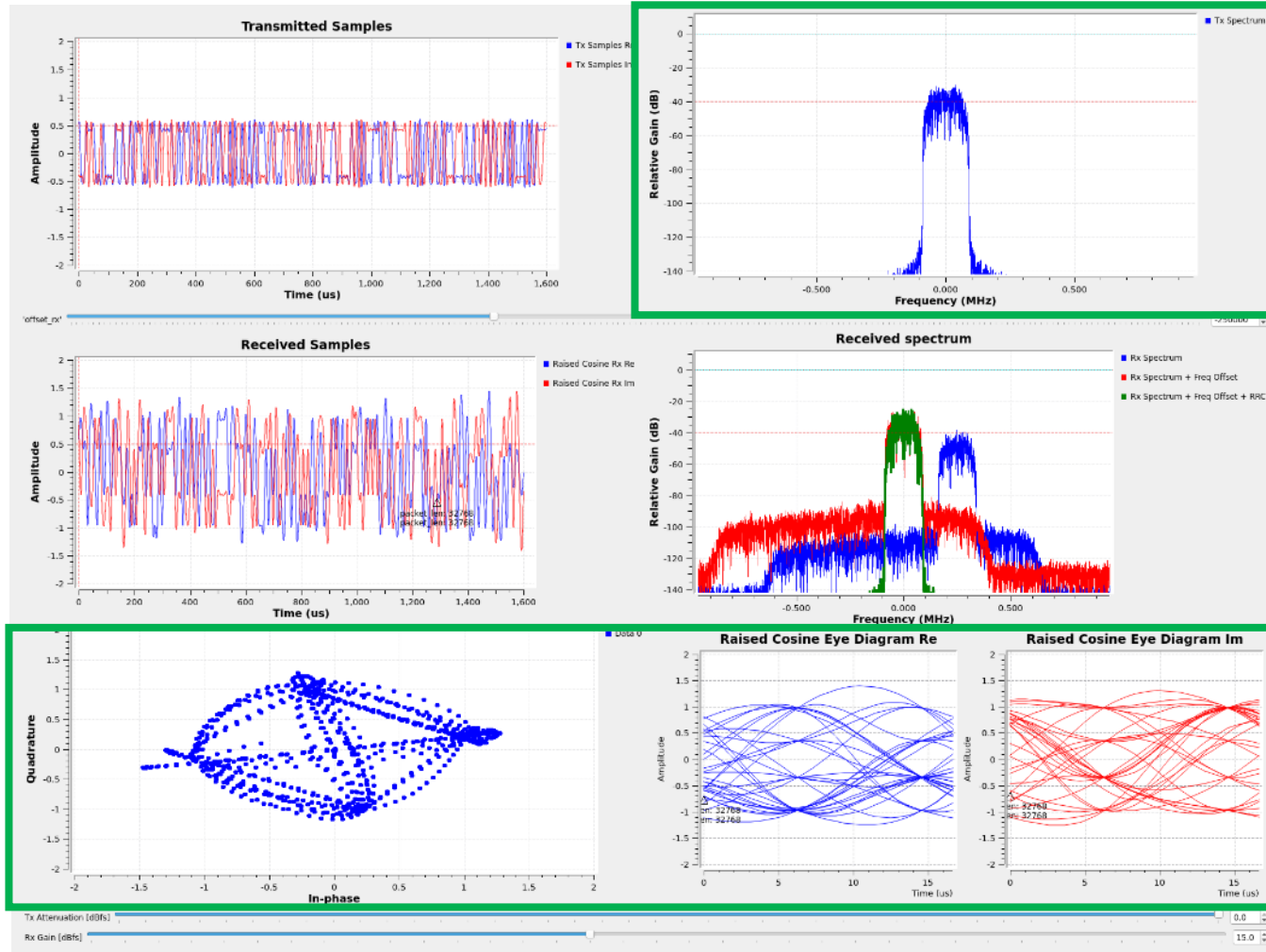


## Receiver

RRC filter at RX



# 12. QPSK – Constellation Modulator in GNU Radio



The BW is optimized due to matched Filters (one from Const. Modulator block at TX and one at the RX).

The RX signal is not decimated By selecting the right samples

## 12. QPSK - Constellation Modulator in GNU Radio

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

- In GNU Radio companion app, open from File -> Open:

[/home/analog/Desktop/ftc\\_2024/12\\_constellation\\_modulator\\_loopback\\_gnuradio/constellation\\_modulator\\_loopback\\_jupiter.grc](#)

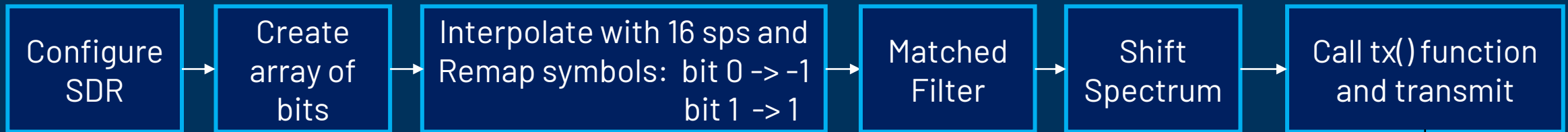
## 12. QPSK – Constellation Modulator in GNU Radio

### ► Conclusion:

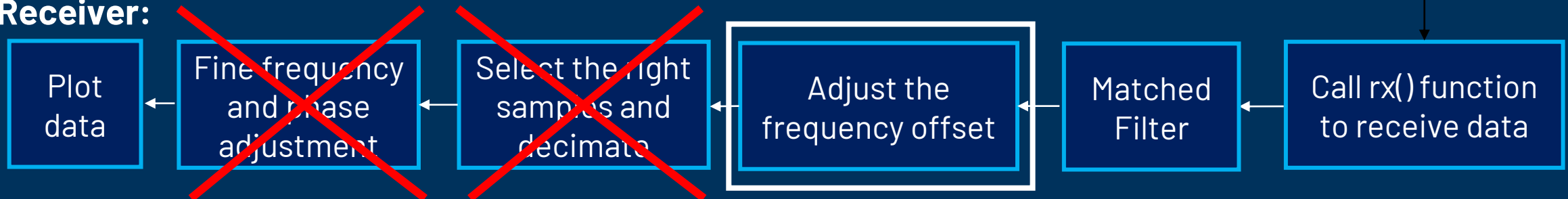
- Using the Constellation Modulator block, the transmitted bits are mapped into IQ symbols and a RRC filter is applied on the Transmitter path, thus the bandwidth used is optimized

# 13. QPSK – Frequency Locked Loop in GNU Radio

## ► Transmitter:



## ► Receiver:



**MISSING**

**ADDED**



## 13. QPSK – Frequency Locked Loop in GNU Radio

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

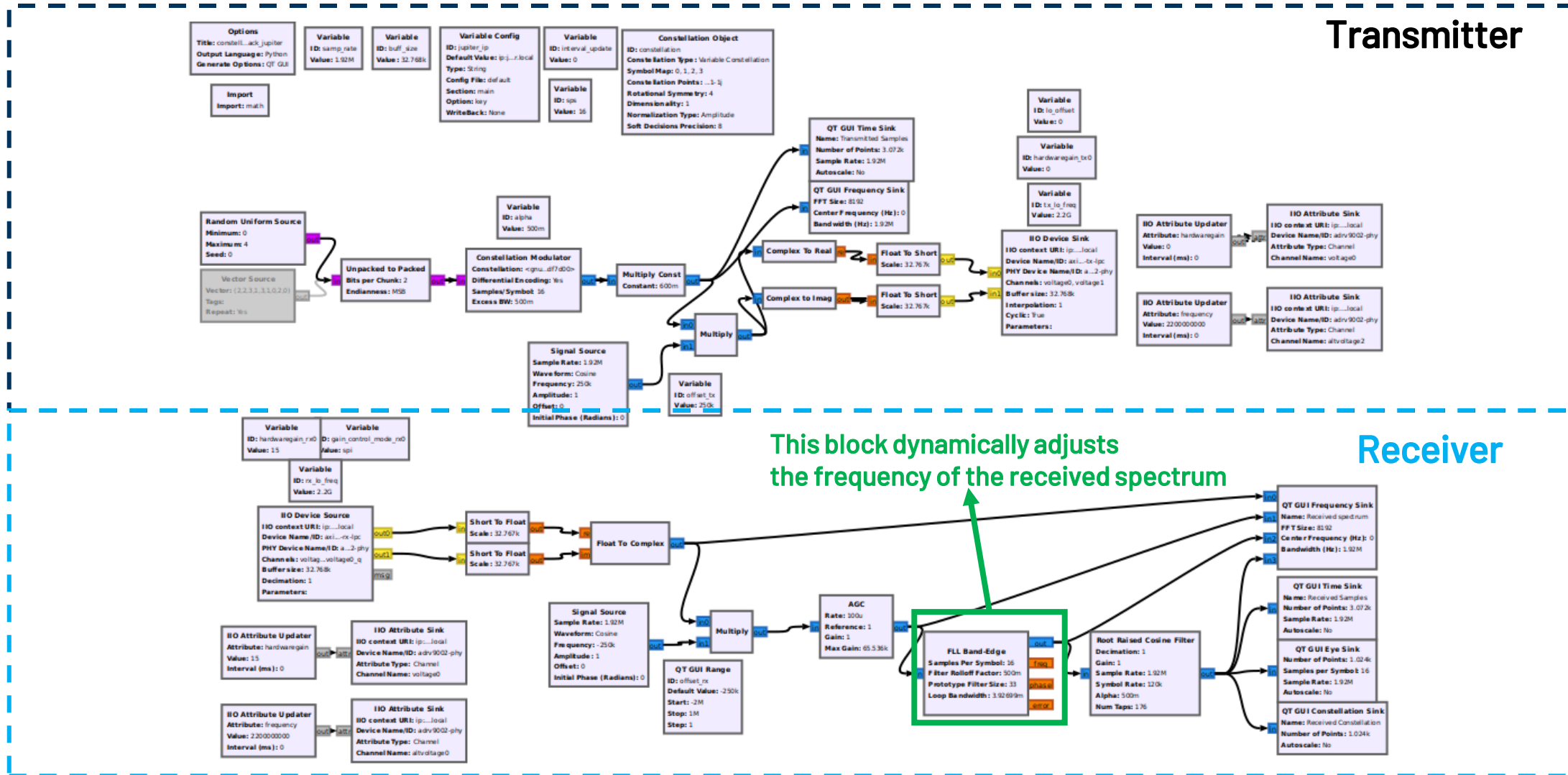
```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

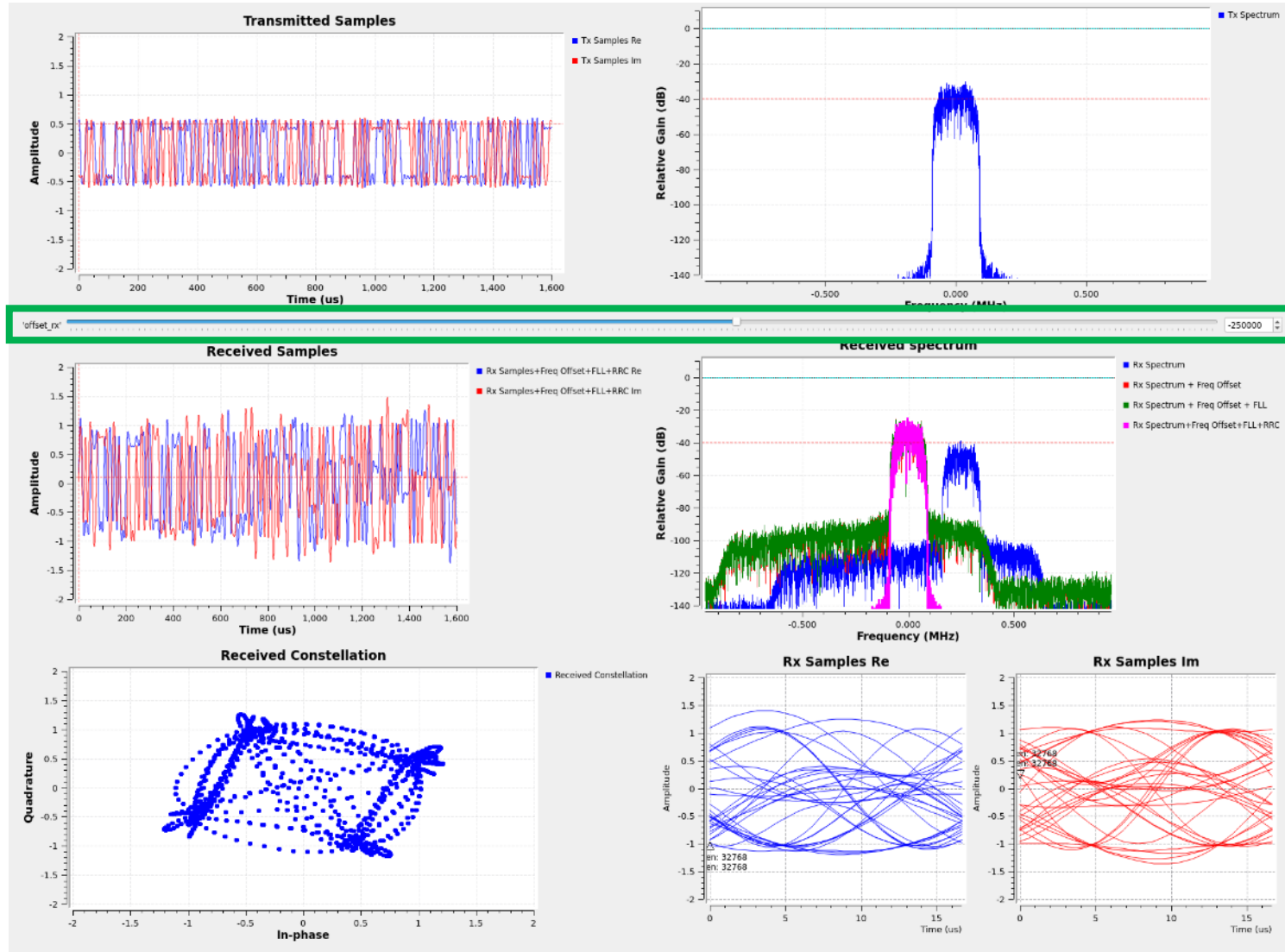
- In GNU Radio companion app, open from File -> Open:

```
/home/analog/Desktop/ftc_2024/13_fll_loopback_gnuradio/FLL_loopback_jupiter.grc
```

# 13. QPSK – Frequency Locked Loop in GNU Radio



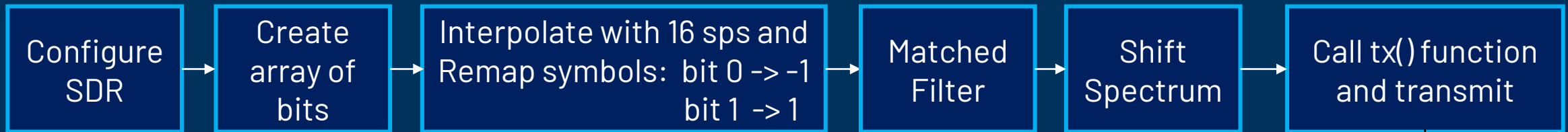
# 13. QPSK – Frequency Locked Loop in GNU Radio



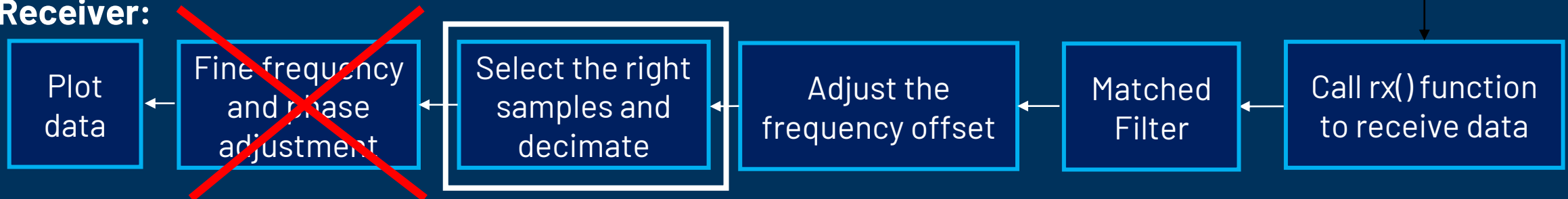
Use this slider to tweak the frequency offset between TX and RX. Observe how the spectrum after FLL block stays centered around DC. A more precise frequency offset Correction still needs to be applied

# 14. QPSK – Symbol Sync in GNU Radio

## ► Transmitter:



## ► Receiver:



**MISSING**

**ADDED**

## 14. QPSK – Symbol Sync in GNU Radio

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

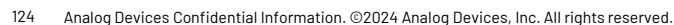
```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

- In GNU Radio companion app, open from File -> Open:

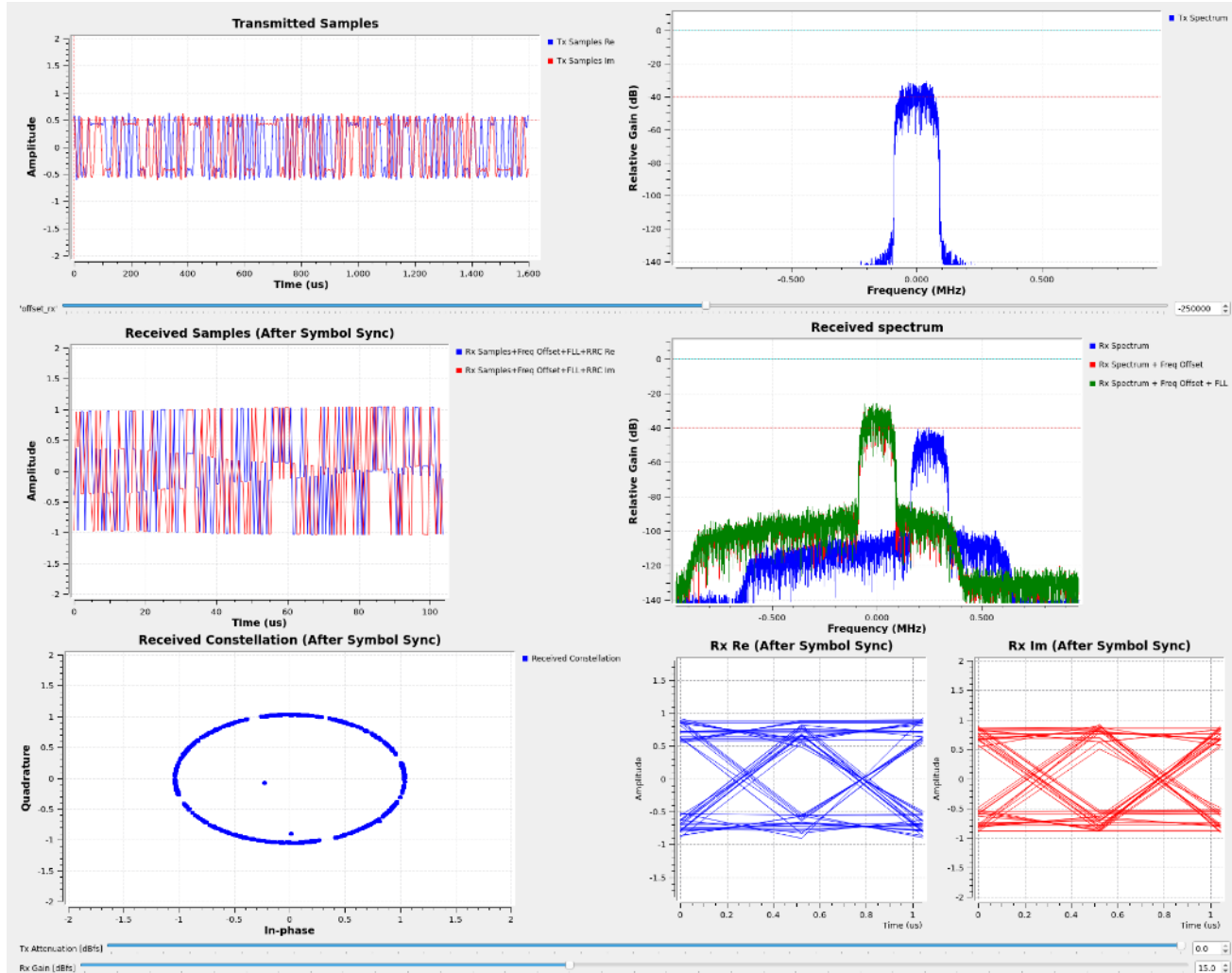
```
/home/analog/Desktop/ftc_2024/14_symbol_sync_loopback_gnuradio/symbol_sync_loopback_jupiter.grc
```





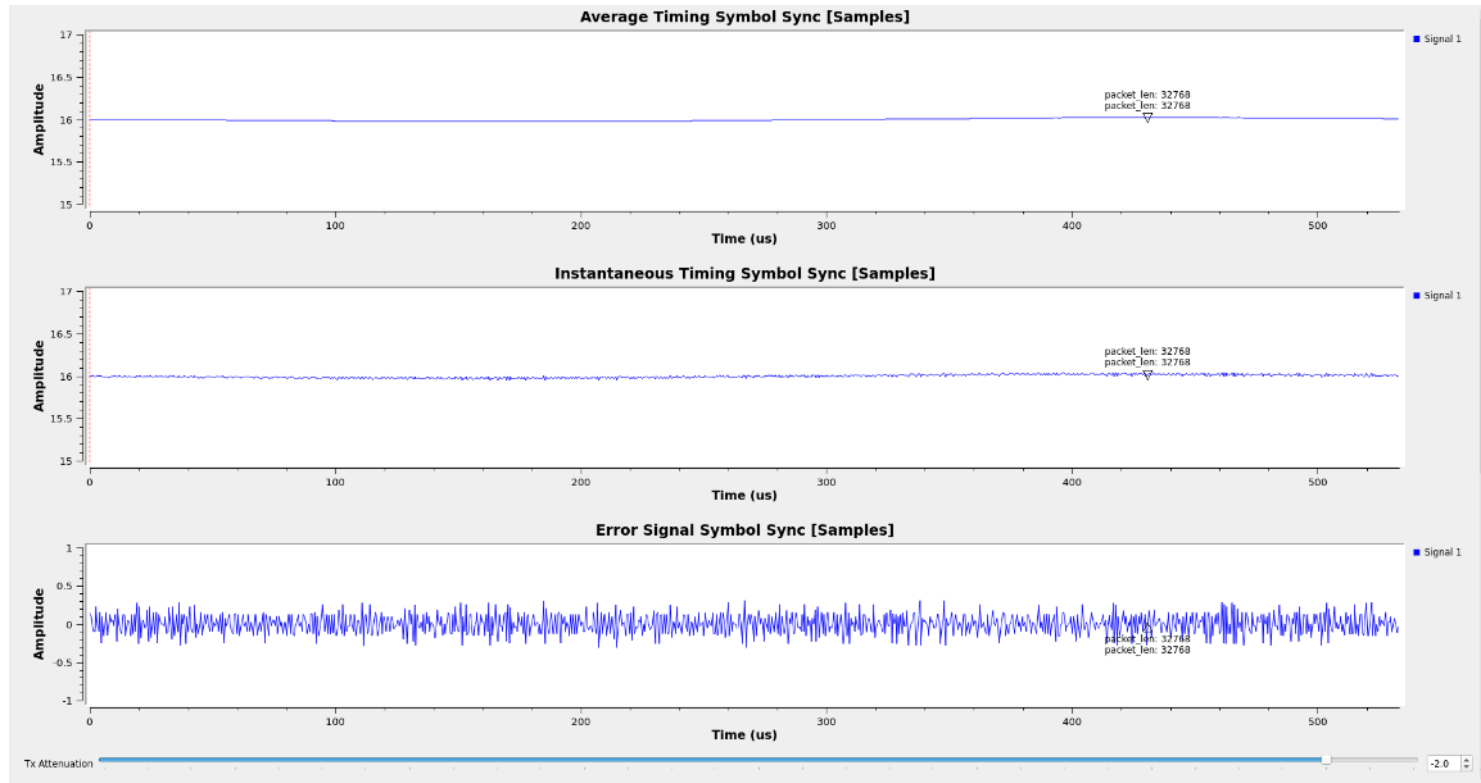


# 14. QPSK – Symbol Sync in GNU Radio



Observe that now all the symbols have the same amplitude on the constellation plot but the imaginary and real parts of the data are still varying due to a remaining frequency and phase offset.

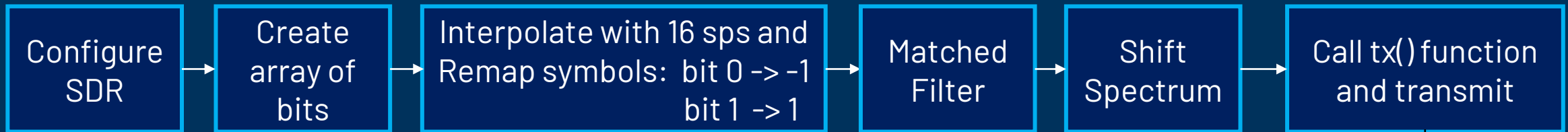
# 14. QPSK – Symbol Sync in GNU Radio



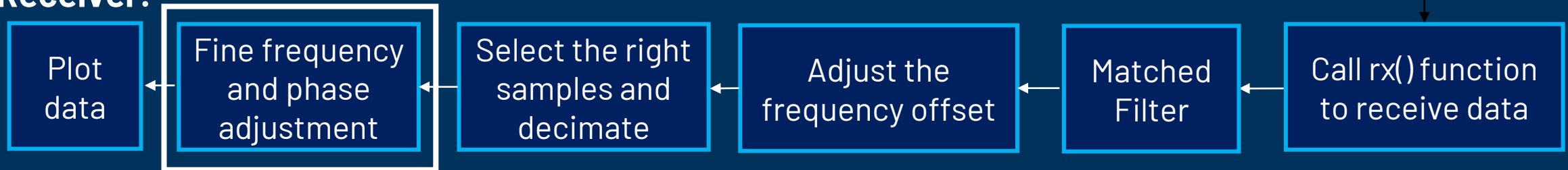
Here are plotted the main dynamic parameter of the symbol sync block. Observe that the timing stays around 16 = sps setting and the error stays around 0.

# 15. QPSK – Costas Loop in GNU Radio

## ► Transmitter:



## ► Receiver:



**ADDED**

## 15. QPSK – Costas Loop in GNU Radio

- Connect Rx and Tx using the SMA cable from the kit by making a **loopback** between RX1 and TX1.



- To open GNU Radio run the following commands:

```
$ cd /home/analog/Desktop
```

```
$ sudo ./start-grc.sh
```

```
analog@analog: ~/Desktop
File Edit Tabs Help
analog@analog:~ $ cd Desktop/
analog@analog:~/Desktop $ sudo ./start-grc.sh
```

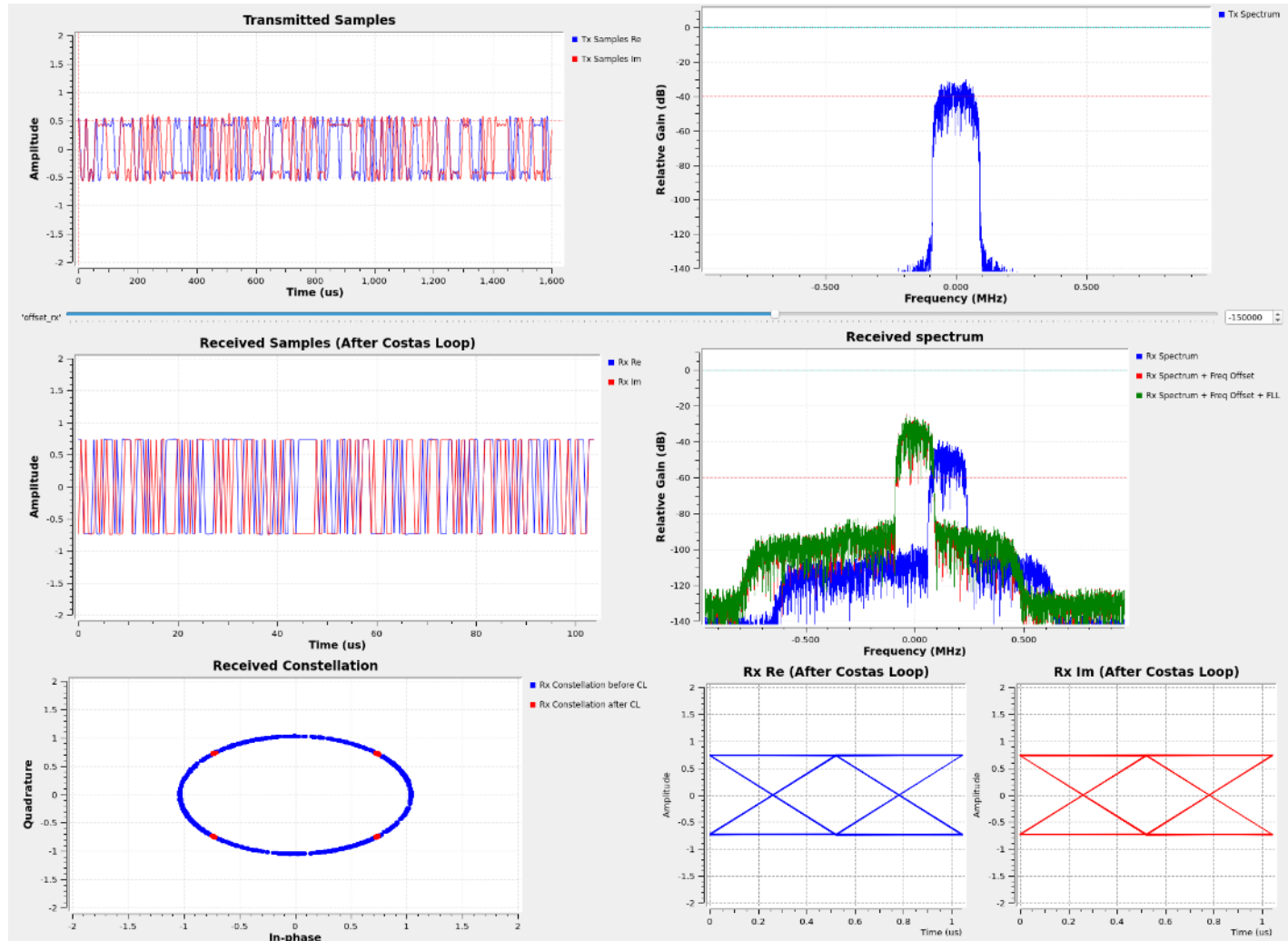
- In GNU Radio companion app, open from File -> Open:

**/home/analog/Desktop/ftc\_2024/15\_costas\_loop\_loopback\_gnuradio/costas\_loop\_loopback\_jupiter.grc**

The diagram illustrates a system-level block diagram for a 5G NR system. It shows a complex signal flow from input variables through various processing blocks like IQ Modulation, FFT, and Channel, to output variables like IQ Demodulation and FFT. It includes multiple feedback loops and a large 'Coded Data' block at the bottom right.

**Costas Loop block: makes fine tuning**  
**➔ to the remaining frequency and phase error.**

# 15. QPSK – Costas Loop in GNU Radio

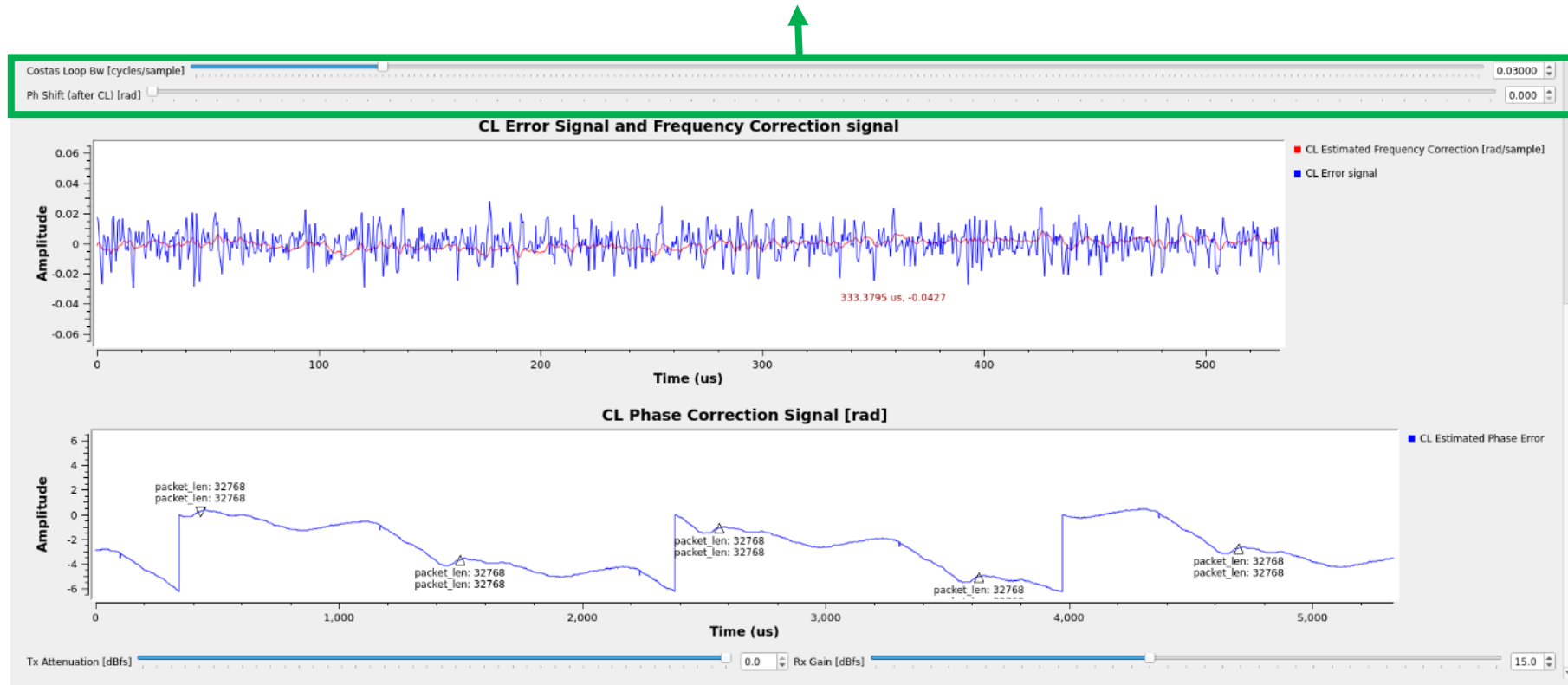


Observe how the symbols are now separated as we wanted and the transitions in the Eye diagrams are less noisy.



# 15. QPSK – Costas Loop in GNU Radio

Here you can tweak the bandwidth of the Costas Loop and see how it influences the error signal and the constellation plot. You can also add a phase shift in radians after Costas Loop.



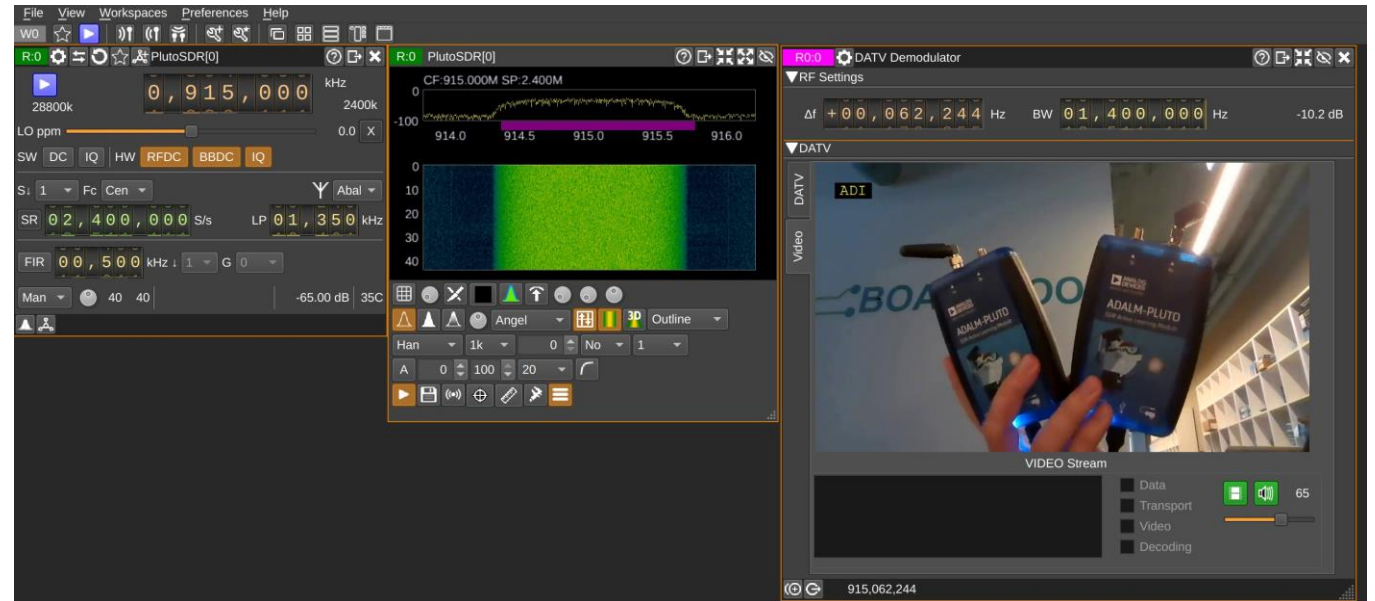
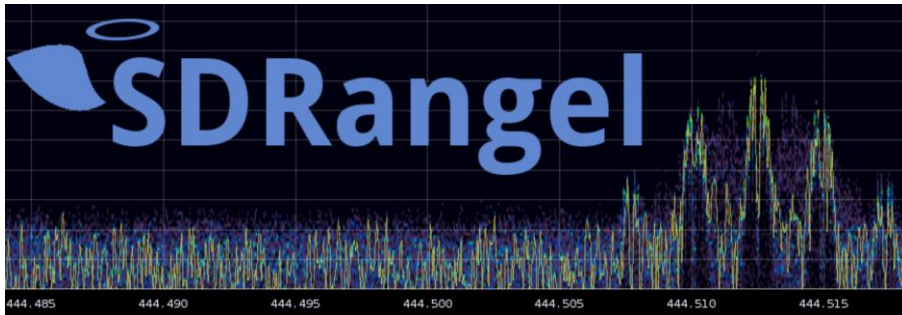
Observe that the error signal is centered around 0 and the Phase Correction signal sits between 0 and  $2\pi$ .

# Other open-source software platforms for SDR

# Other open-source software platforms for SDR

## SDR Angel

<https://www.sdrangel.org/>



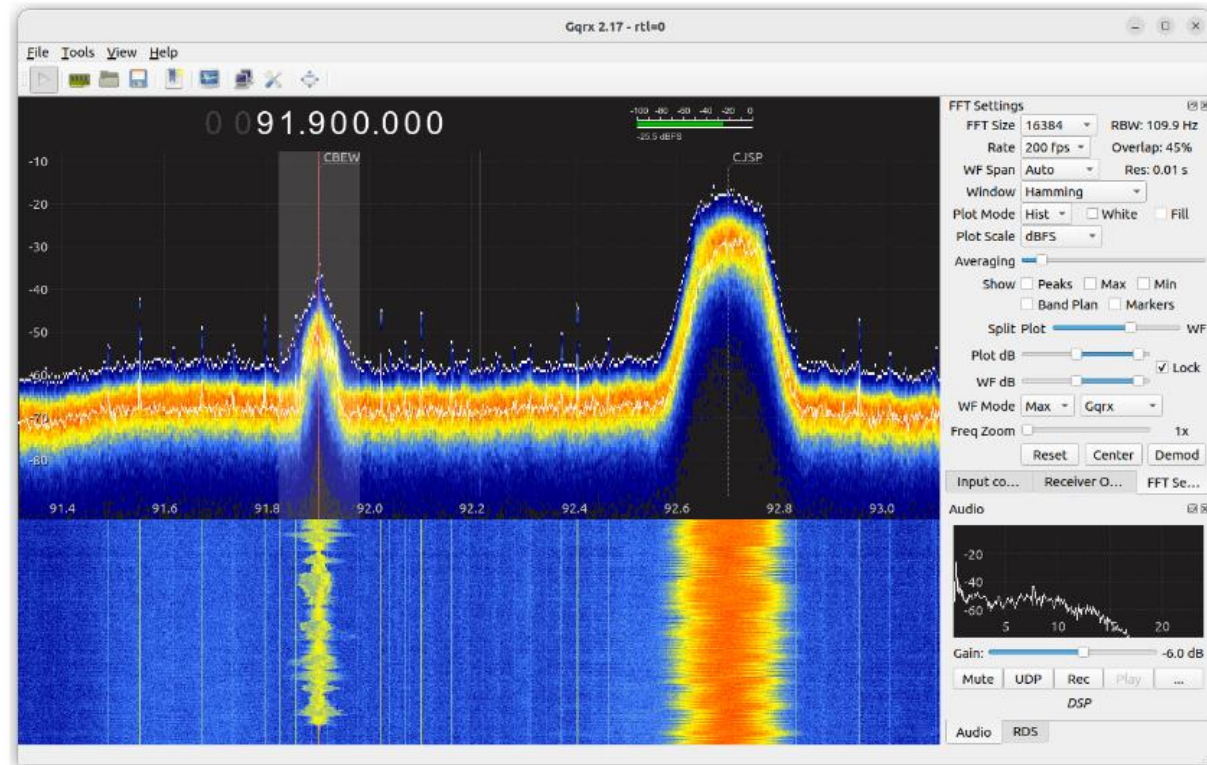
# Other open-source software platforms for SDR

## GQRX

### Welcome to gqrx

Gqrx is an open source software defined radio receiver (SDR) powered by the [GNU Radio](#) and the [Qt](#) graphical toolkit.

<https://www.gqrx.dk/>



# Conclusions

- ▶ **In SDR, a lot of the signal-processing tasks are moved in software, resulting in a faster development time.**
- ▶ **The software is easily portable between our SDRs which allows for a lower cost system for prototyping (such as Pluto).**
- ▶ **There is a lot of open-source software and platforms that can accelerate the development of SDR systems.**
- ▶ **The same SDR hardware can be used in multiple applications that in the past needed separate hardware, resulting in a much lower cost of the system (for prototyping) and development resources.**

# References



- [1] Pluto - <https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm-pluto.html>
- [2] Jupiter - <https://wiki.analog.com/resources/eval/user-guides/jupiter-sdr/hardware-overview>
- [3] Talise SOM - <https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adrv9009-zu11eg.html>
- [4] T. F. Collins, R. Getz, Di Pu, A. M. Wyglinski,- Software-Defined Radio for Engineers,2018, ISBN-13: 978-1-63081-457-1.\
- [5] GNU Radio - <https://www.gnuradio.org/>
- [6] M. Lichtman, "PySDR" - <https://pysdr.org/>
- [7] T. F. Collins - [https://www.gnuradio.org/grcon/grcon18/presentations/ADI\\_Transceivers\\_A\\_Deep\\_Dive/](https://www.gnuradio.org/grcon/grcon18/presentations/ADI_Transceivers_A_Deep_Dive/)
- [8] J. Gallachio - <https://github.com/gallicchio/learnSDR>
- [9] SDRangel - <https://www.sdrangel.org/>
- [10] K. Mueller and M. Muller, "Timing recovery in digital synchronous data receivers," IEEE Trans. Commun., vol. C-24, no. 5, pp. 516–531, May 1976.
- [11] J. Costas, "Synchronous Communications," Proceedings of the IEEE, vol. 44, p. 1713-1718, 1956
- [12] PyADI-II0 - <https://github.com/analogdevicesinc/pyadi-iio/>
- [13] iio\_attr documentation - [https://wiki.analog.com/resources/tools-software/linux-software/libiio/iio\\_attr](https://wiki.analog.com/resources/tools-software/linux-software/libiio/iio_attr)
- [14] All the documentation from this workshop - <https://analogdevicesinc.github.io/documentation/learning/index.html>



# References



[15] Kuiper 2 source code: <https://github.com/analogdevicesinc/adi-kuiper-gen/tree/staging/kuiper2.0>

[16] Kuiper 2 built images: [https://github.com/analogdevicesinc/adi-kuiper-gen/actions/workflows/kuiper2\\_0-build.yml](https://github.com/analogdevicesinc/adi-kuiper-gen/actions/workflows/kuiper2_0-build.yml)



Thank You!

*Please Remember to **Rate** this Session in the Mobile App!*