ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

ADI FTC
COLLABORATE • CREATE • ACCELERATE

Just enough Software and HDL for High Speed designs

Michael Hennerich
Mihai Bancisor

Typical Customer Design Flow

Research
- Behavioral Simulation
- Device Evaluation
  - Meassure: SFDR, SNR, EVM, NF, NSD, etc.

Algorithm Development
- MATLAB/Python/GRC reference implementation
- Hardware streaming

Design Elaboration
- Modeling
  - MATLAB/Python/GRC
- Hardware streaming
- Data type conversion

Prototype
- Deployment to development board
- Design optimization
- HDL Integration
- Driver Integration

Production
- Deployment to custom hardware
- Validation with complete hardware solution

PlutoSDR/JupiterSDR

Evaluation board FMC + FPGA Carrier of choice

Full Custom design using same HDL/SW/Infrastructure

Software in the Design-in Journey

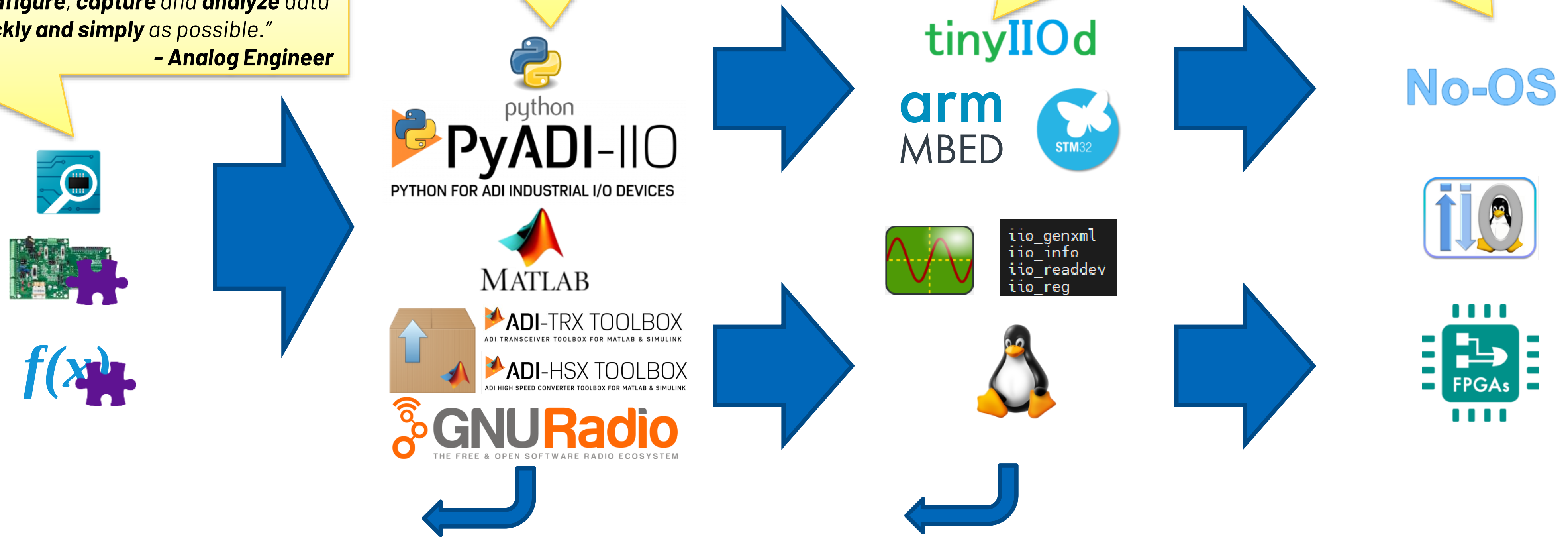
ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

"I need to **configure**, **capture** and **analyze** data as **quickly and simply** as possible."
- Analog Engineer

"I want to **model and process data** using familiar tools."
- Modelling/Domain Expert

"I want to **target** more **representative hardware and software configurations**."
- Embedded Software Engineer

"I need to **implement** the **software** and/or **digital interface** with my host MCU/FPGA"
- Embedded Software & Digital HDL Engineers


f(x)
python
PyADI-IIO
PYTHON FOR ADI INDUSTRIAL I/O DEVICES
MATLAB
ADI-TRX TOOLBOX
ADI TRANSCEIVER TOOLBOX FOR MATLAB & SIMULINK
ADI-HSX TOOLBOX
ADI HIGH SPEED CONVERTER TOOLBOX FOR MATLAB & SIMULINK
GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM
tinyIIOd
arm MBED
STM32
No-OS
FPGAs

```
iio_genxml
iio_info
iio_readdev
iio_reg
```

Evaluation & Research
Evaluation Board,
Software Model*

Data Capture, Analysis, Algorithm Development
3rd Party Tool Support

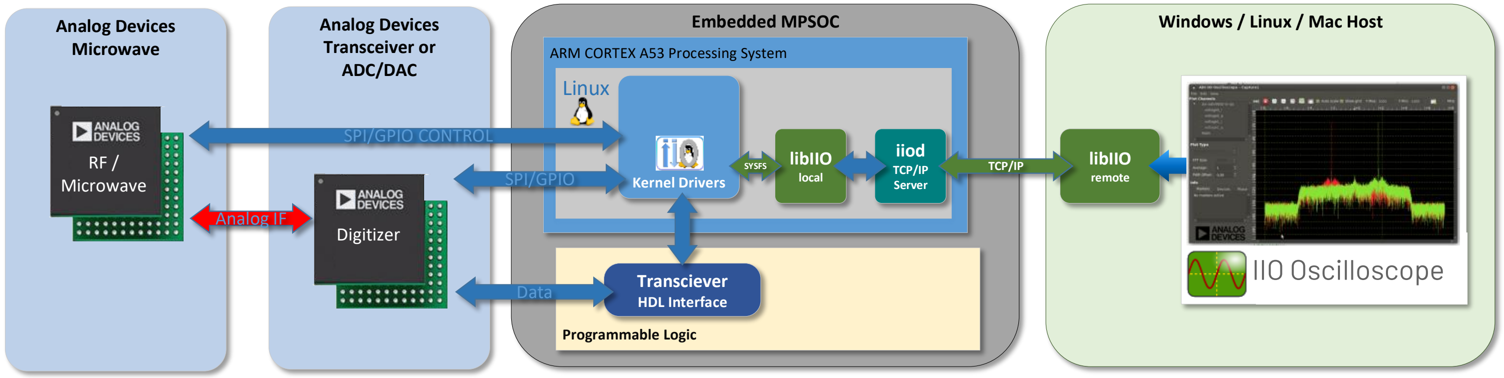
Prototyping
Firmware Applications & Linux Support

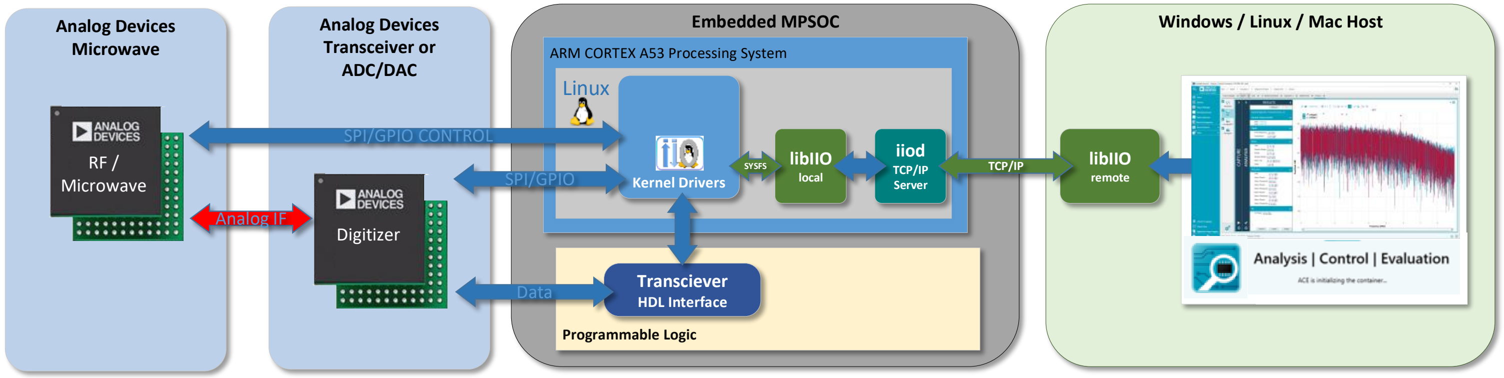
Development & Production
No-OS & Linux drivers & HDL

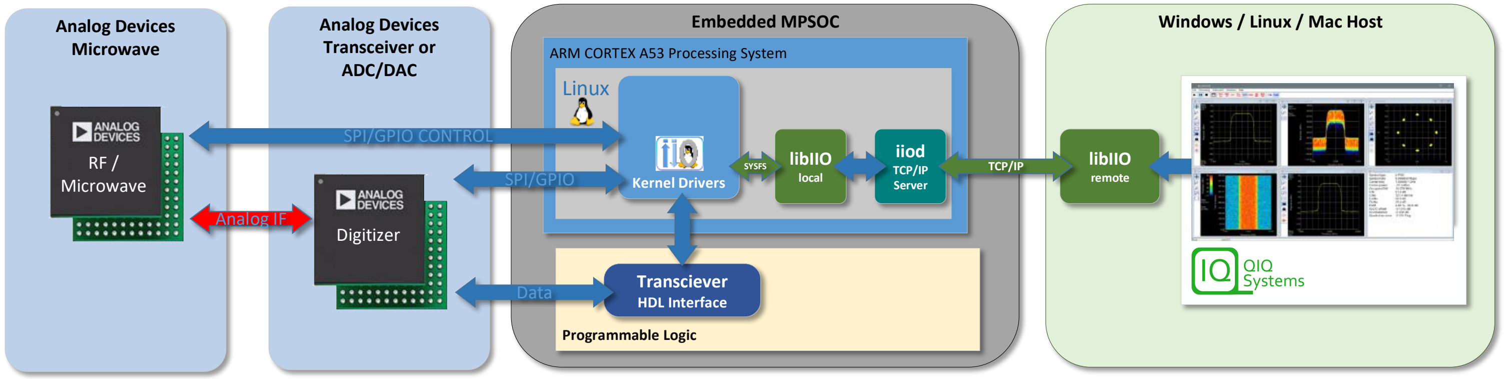
** Software Model is optional*

Increasing Technical Expertise & Time

Evaluation, Test and Analysis



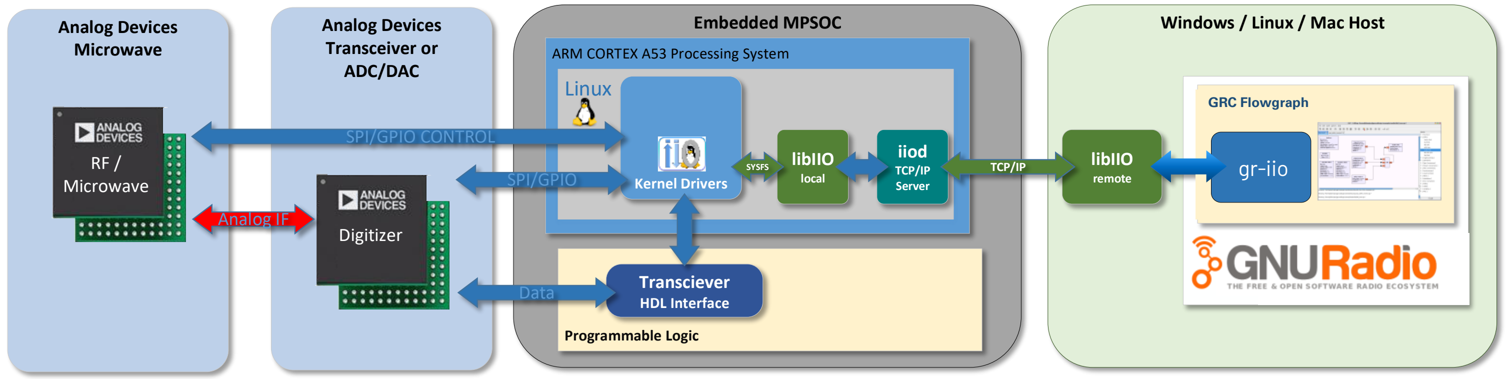


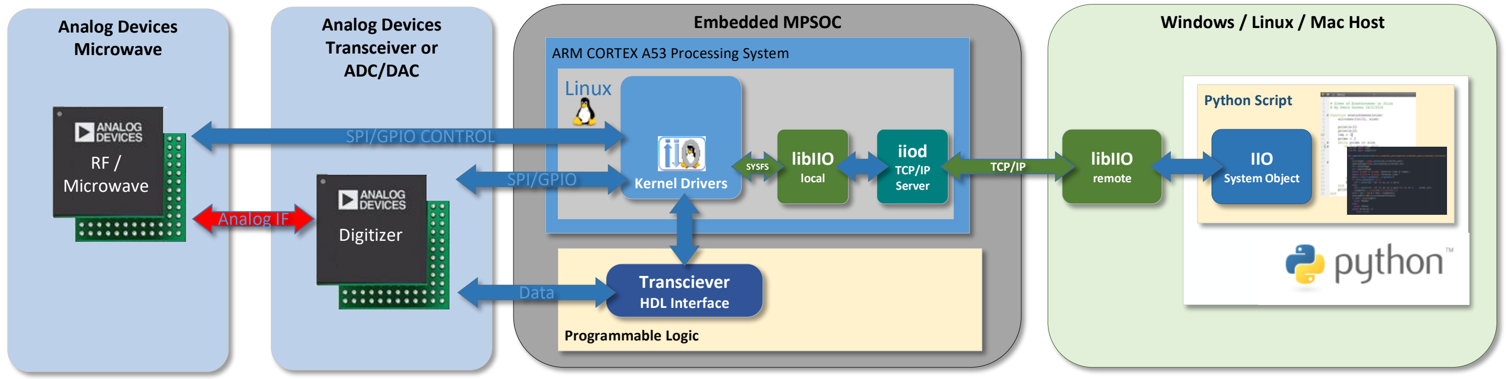


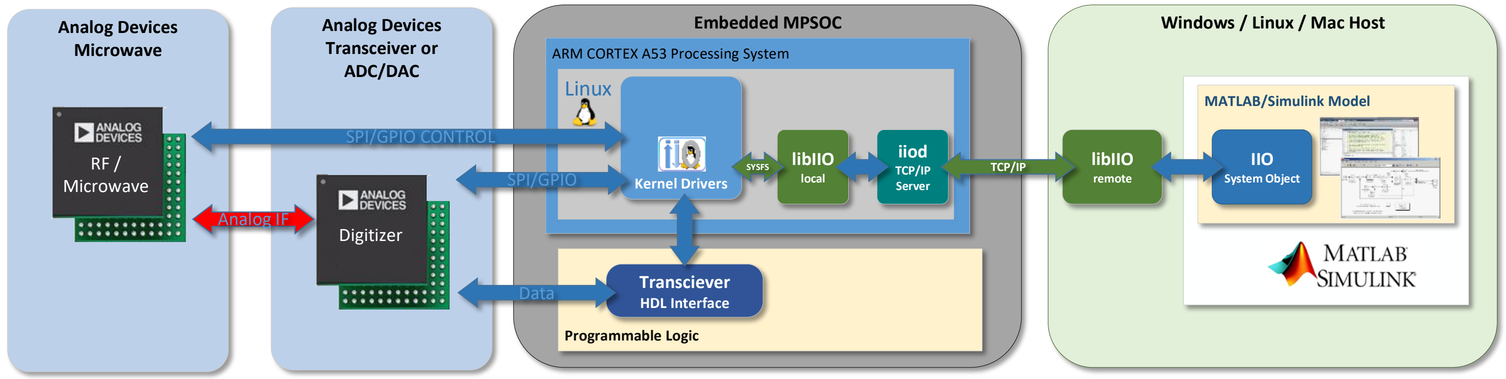
Single cohesive software solution – meeting customers in their ecosystem or at their tools of choice

- ❑ **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

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

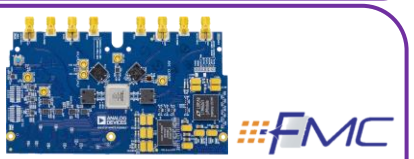
Hardware Development

Evaluation/Prototyping

Open Market Development platforms (Off the shelf carrier boards)

ADI Evaluation boards (Daughter Boards)

FMC Compatible Dev platforms



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

SW, Infrastructure & Tools

Open Source & Bare Metal Device level Drivers

HDL/FPGA Reference Designs

ADI-KUIPER-LINUX
LINUX DEVICE DRIVERS FOR ADI PERIPHERALS

ADI-JESD204
ADI JESD204 INTERFACE FRAMEWORK

Matlab Toolbox Support

ADI LibIIO

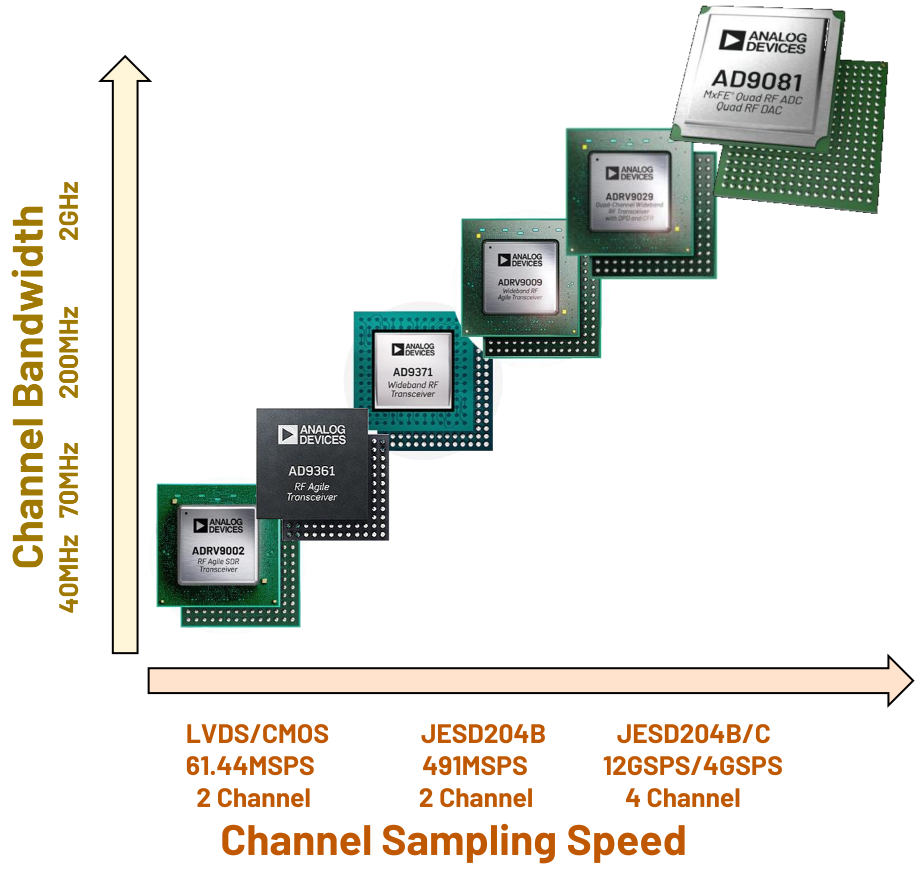
ADI IIO-Scope

PyADI-JIF
JESD204 FRAMEWORK CONFIGURATOR

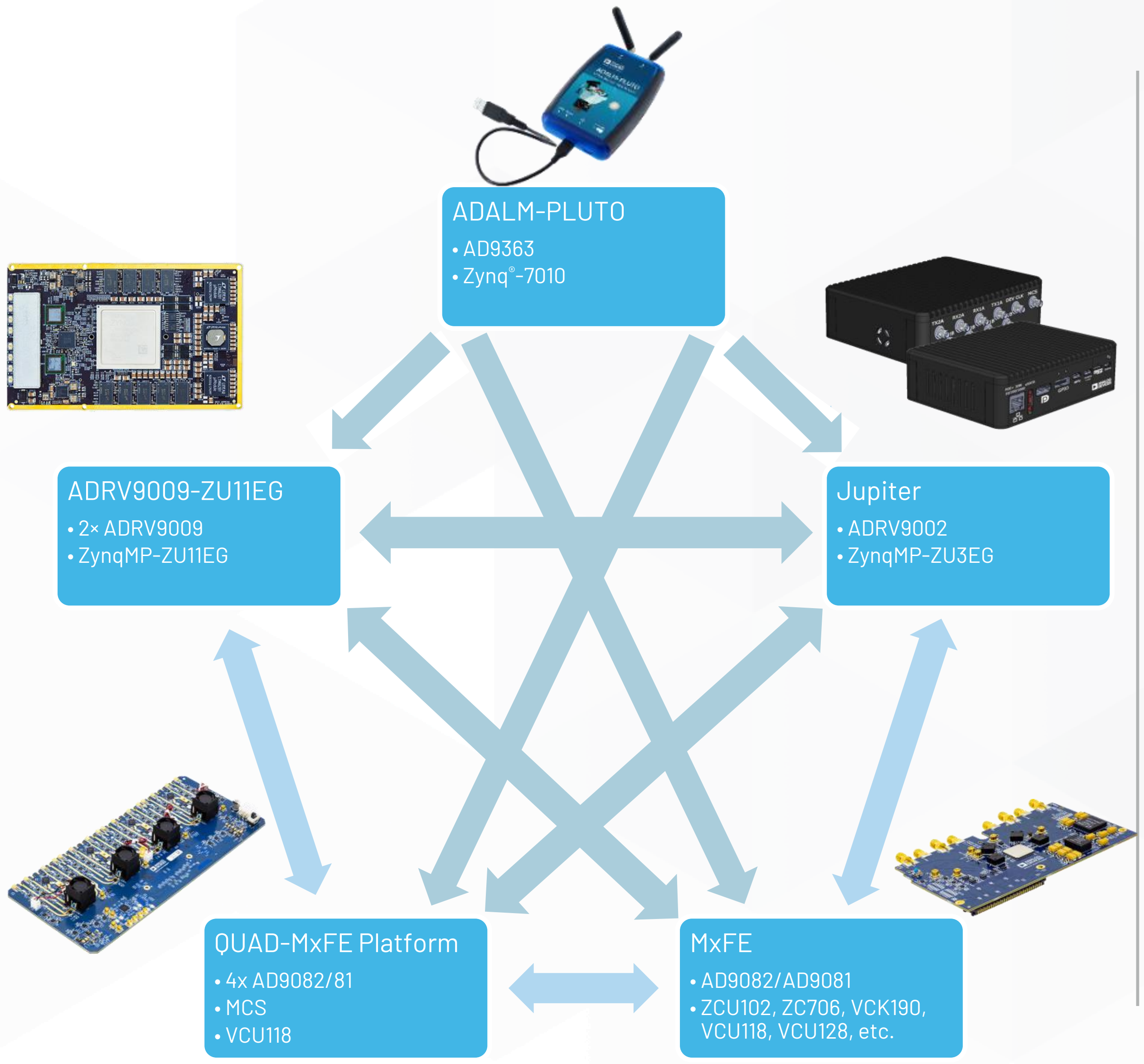
MATLAB python

PyADI-IIO
PYTHON FOR ADI INDUSTRIAL I/O DEVICES

Options for different RF Applications


Channel Bandwidth
40MHz
70MHz
200MHz
2GHz
ADRV9002 RF Agile SDR Transceiver
AD9361 RF Agile Transceiver
AD9371 Wideband RF Transceiver
ADRV9009 Wideband RF Agile Transceiver
ADRV9029
AD9081 MxFE Quad RF ADC Quad RF DAC
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



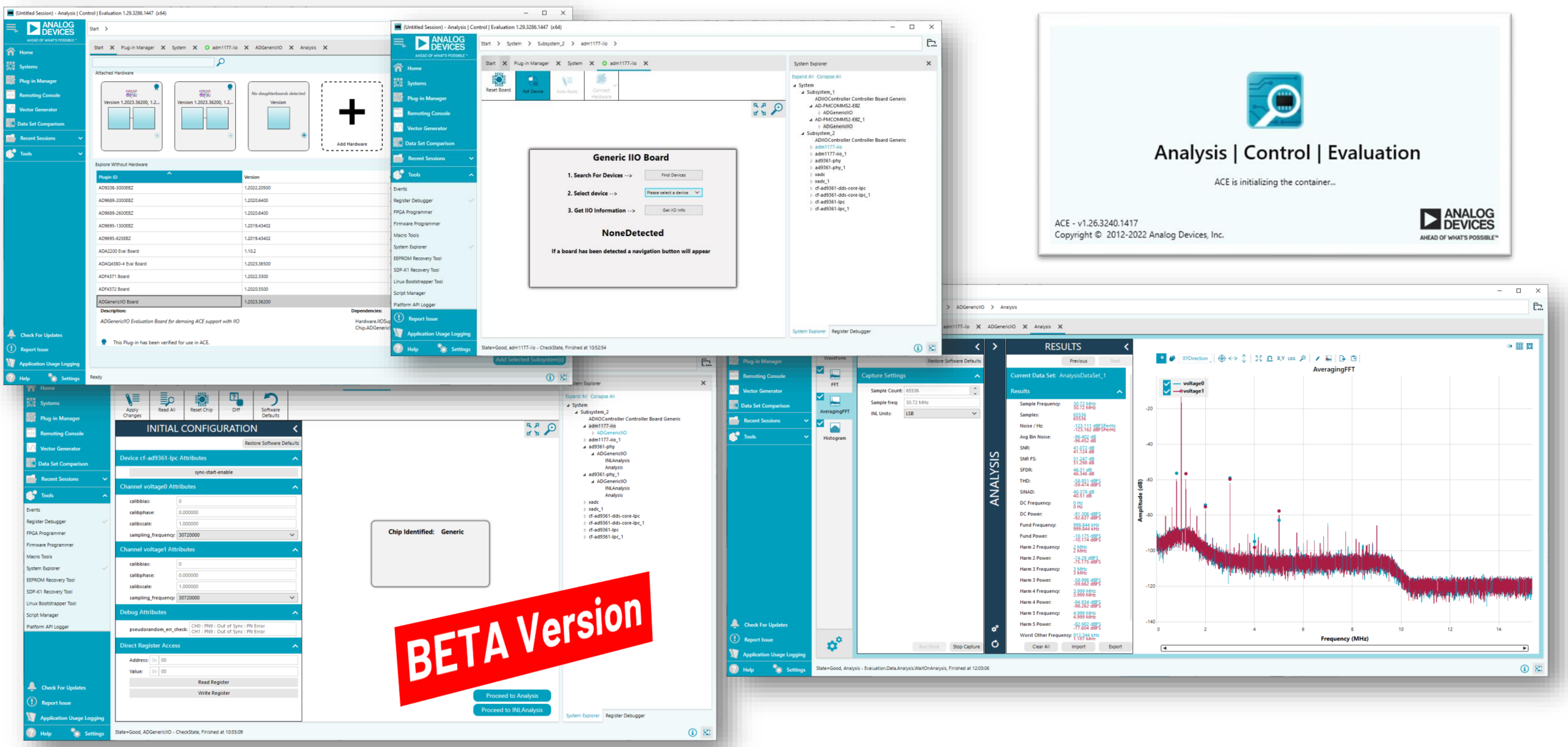
- Shares same software/HDL/hardware stack
  - Makes it easy to move from one to the other
  - Differentiated on form factor, number of channels, connectivity, expandability, FPGA resources, CPU resources
- Start with ADALM-PLUTO
  - Stream to MATLAB®, Simulink®, or GNU Radio via USB
  - Take data in 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
  - Common Linux® kernel at github.com/analogdevicesinc/linux

*"I need to **configure**, **capture** and **analyze** data as **quickly and simply** as possible."*
- Analog Engineer

Analysis | Control | Evaluation

- (ACE) Software
- IIO Oscilloscope
- Scopy
- qIQ Receiver

ACE ADGenericIIO Board Plugin



IIO-Scope

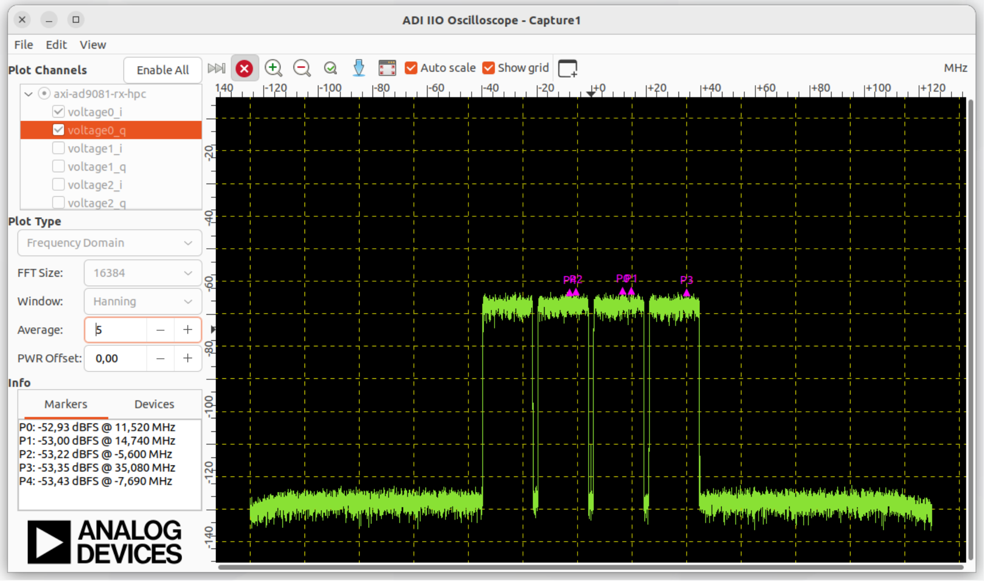
- Capture and display data
  - Time domain (with trigger support)
  - Frequency domain
  - Constellation plot
  - Markers
  - Math operations
- Device configuration
- Plug-in system allow to create device or complex specialized GUI
- Should support **any** IIO device
- Cross platform

Analog Devices Inc. Industrial I/O Oscilloscope

HEAD-g938d5a0

Homepage

Copyright Analog Devices, 2012-2022


ADI IIO Oscilloscope - Capture1

Plot Type: Frequency Domain

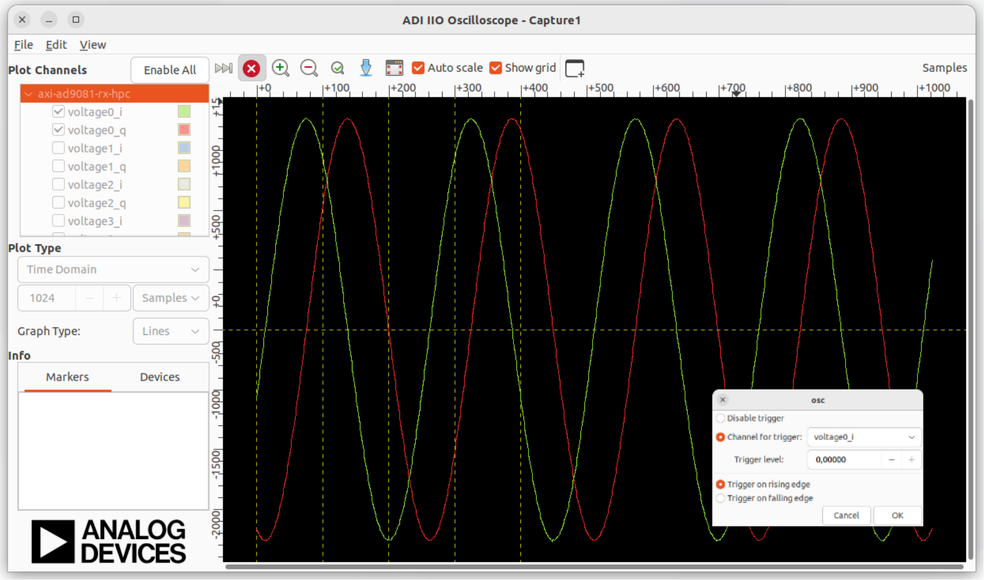
FFT Size: 16384

Window: Hanning

Average: 5

PWR Offset: 0,00

P0: -52,93 dBFS @ 11,520 MHz
P1: -53,00 dBFS @ 14,740 MHz
P2: -53,22 dBFS @ -5,600 MHz
P3: -53,35 dBFS @ 35,080 MHz
P4: -53,43 dBFS @ -7,690 MHz


ADI IIO Oscilloscope - Capture1

Plot Type: Time Domain

1024 Samples

Graph Type: Lines

osc

Disable trigger

Channel for trigger: voltage0_i

Trigger level: 0,00000

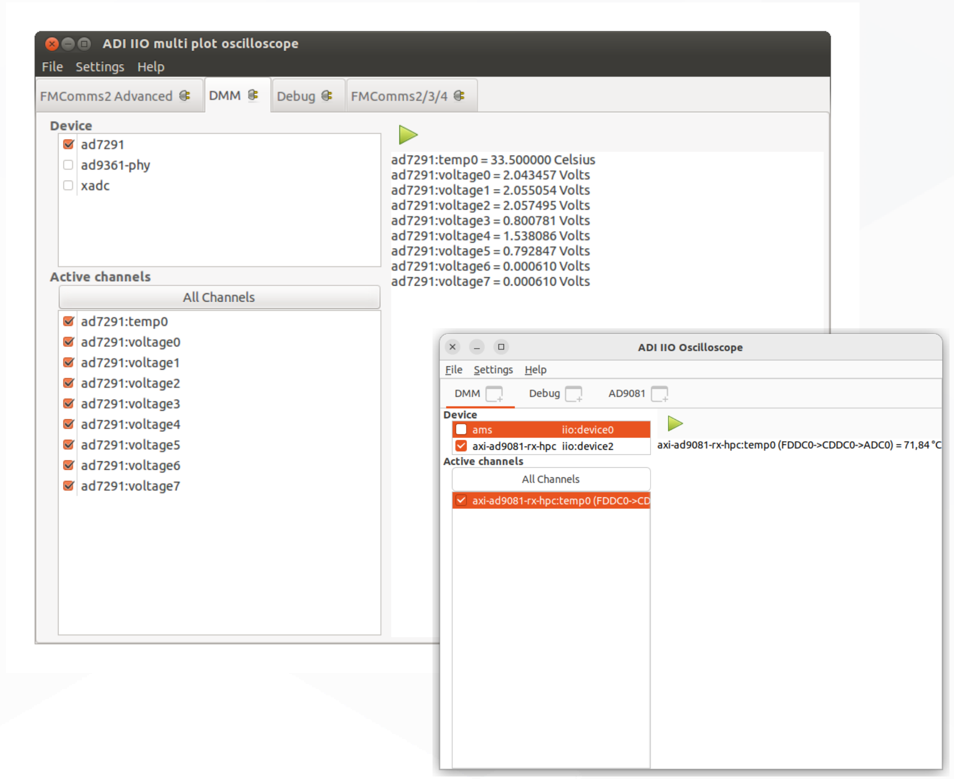
Trigger on rising edge

Trigger on falling edge

Generic plugins

▶ DMM Plugin

- The Digital Multimeter continuously displays device specific data once the start button is activated



ADI IIO multi plot oscilloscope

File Settings Help

FMComms2 Advanced | DMM | Debug | FMComms2/3/4

Device
- ☑ ad7291
- ☐ ad9361-phy
- ☐ xadc

Active channels

All Channels
- ☑ ad7291:temp0
- ☑ ad7291:voltage0
- ☑ ad7291:voltage1
- ☑ ad7291:voltage2
- ☑ ad7291:voltage3
- ☑ ad7291:voltage4
- ☑ ad7291:voltage5
- ☑ ad7291:voltage6
- ☑ ad7291:voltage7

```
ad7291:temp0 = 33.500000 Celsius
ad7291:voltage0 = 2.043457 Volts
ad7291:voltage1 = 2.055054 Volts
ad7291:voltage2 = 2.057495 Volts
ad7291:voltage3 = 0.800781 Volts
ad7291:voltage4 = 1.538086 Volts
ad7291:voltage5 = 0.792847 Volts
ad7291:voltage6 = 0.000610 Volts
ad7291:voltage7 = 0.000610 Volts
```

ADI IIO Oscilloscope

File Settings Help

DMM | Debug | AD9081

Device
- ☐ ams iio:device0
- ☑ axi-ad9081-rx-hpc iio:device2

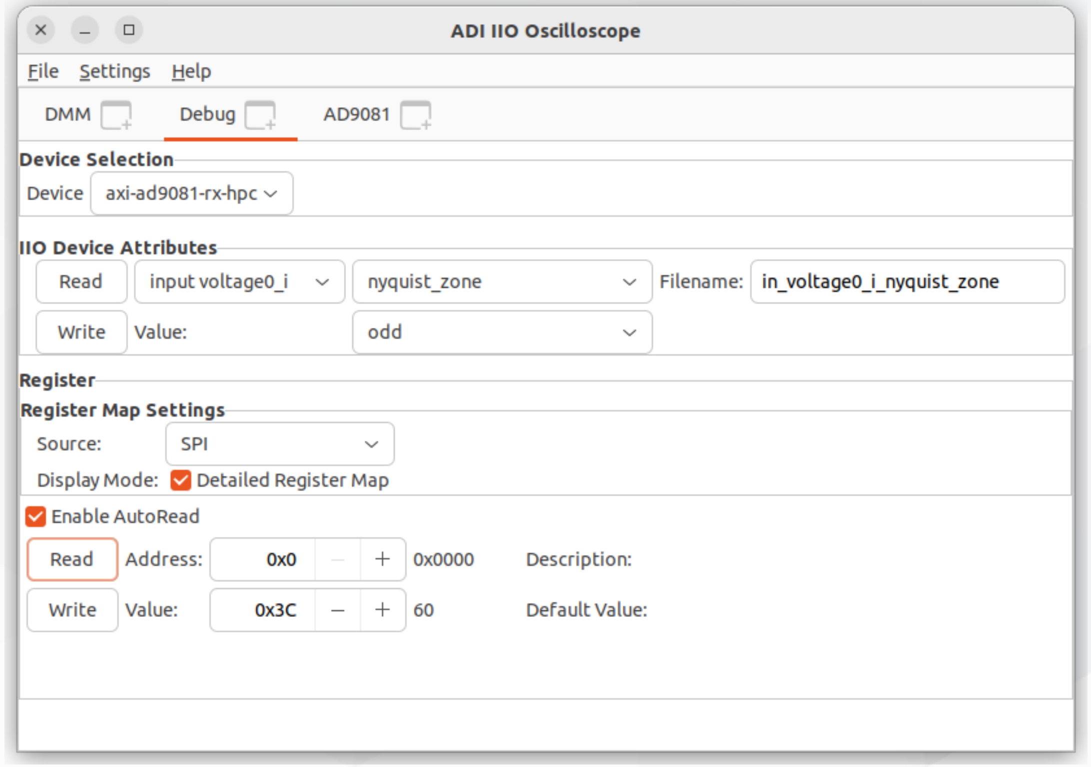
Active channels

All Channels
- ☑ axi-ad9081-rx-hpc:temp0 (FDDC0->CD

```
axi-ad9081-rx-hpc:temp0 (FDDC0->CDDC0->ADC0) = 71,84 °C
```

▶ Debug Tab/Plugin

- Tool for device debugging and control
- Read Write device/channel attributes and low-level registers



ADI IIO Oscilloscope

File Settings Help

DMM | Debug | AD9081

Device Selection

Device: axi-ad9081-rx-hpc

IIO Device Attributes

Read | input voltage0_i | nyquist_zone | Filename: in_voltage0_i_nyquist_zone

Write | Value: | odd

Register

Register Map Settings

Source: SPI

Display Mode: ☑ Detailed Register Map

☑ Enable AutoRead

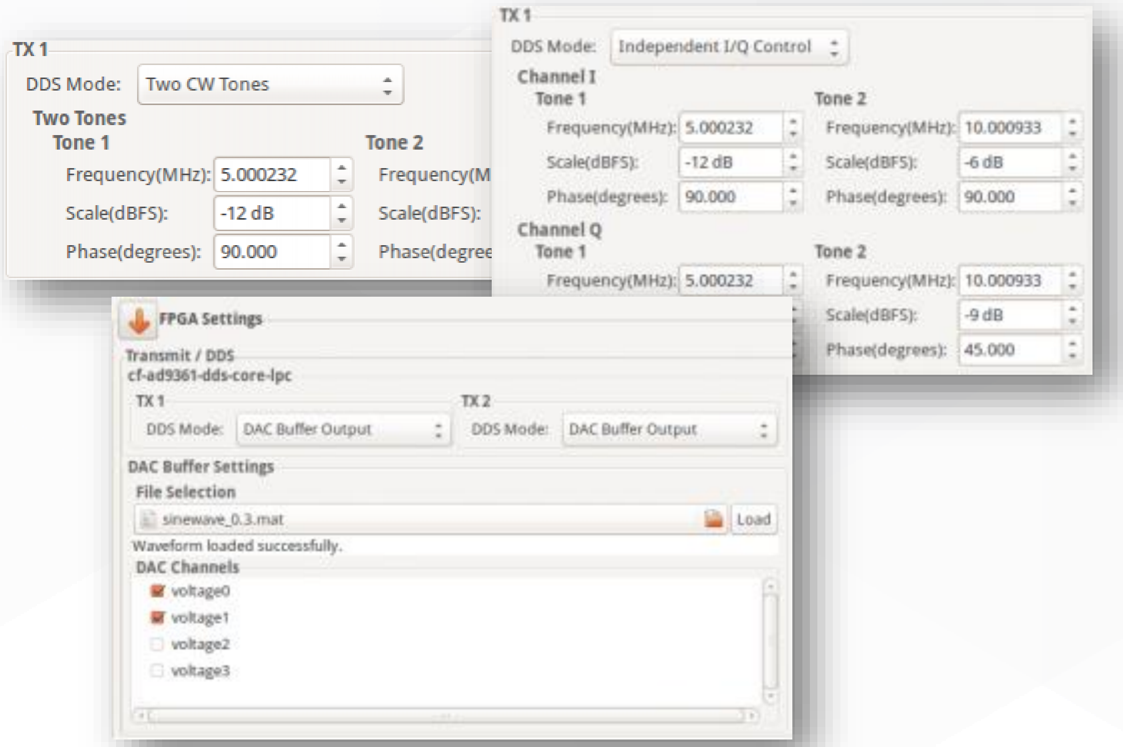
Read | Address: 0x0 | 0x0000 | Description:

Write | Value: 0x3C | 60 | Default Value:

Specific control plugins

FPGA Settings

- The plugin provides several options on how the transmitted data is generated
- Two tone **Direct Digital Synthesizer** (DDS) to transmit a bi-tonal signal on channels I and Q of the DAC. Or it is possible to use the **Direct Memory Access (DMA)** facility to transmit custom data that you have stored in a file.

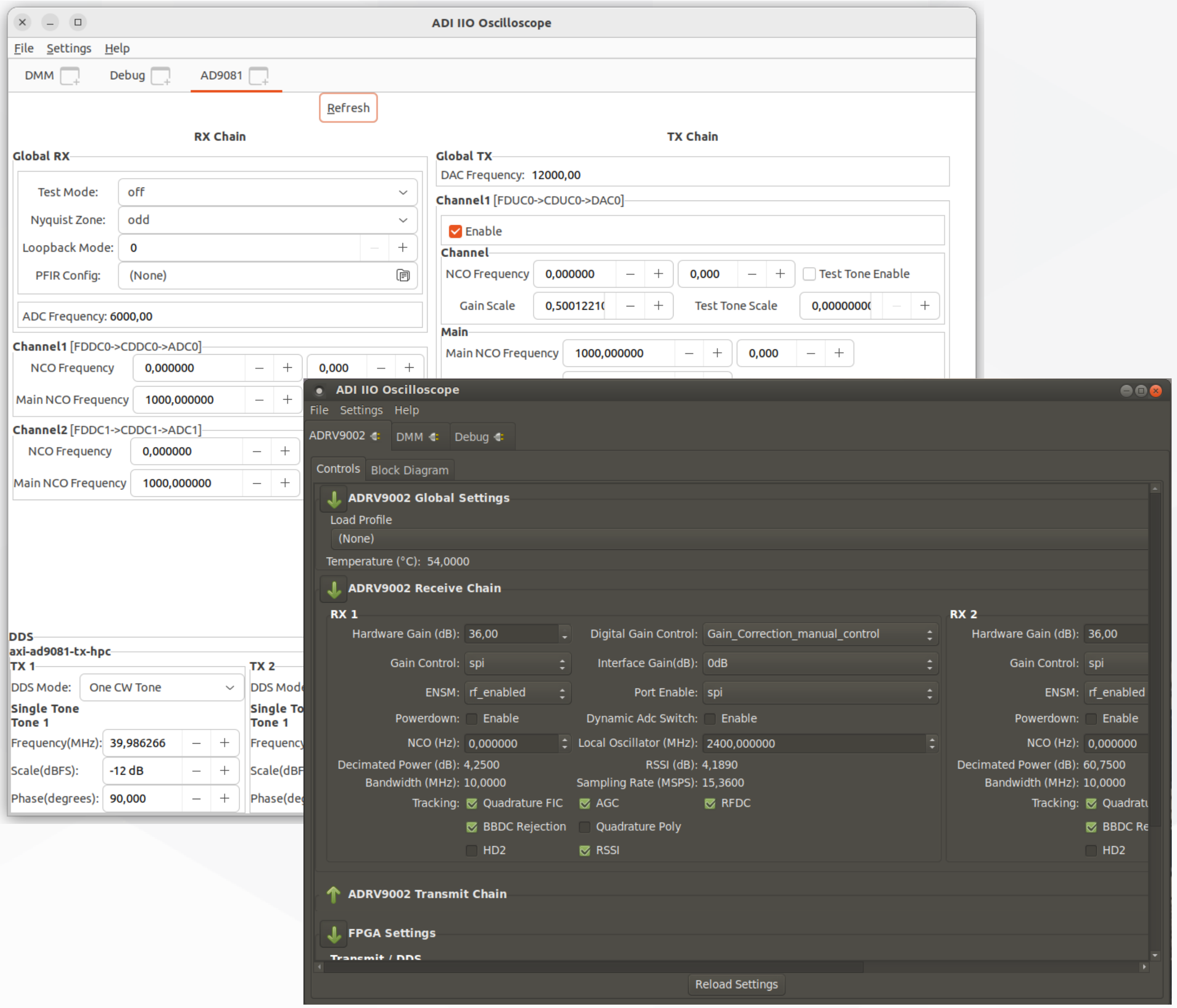


TX 1
DDS Mode: Two CW Tones
Two Tones
Tone 1: Frequency(MHz): 5.000232; Scale(dBFS): -12 dB; Phase(degrees): 90.000
Tone 2: Frequency(M…; Scale(dBFS):; Phase(degree…

TX 1
DDS Mode: Independent I/Q Control
Channel I
Tone 1: Frequency(MHz): 5.000232; Scale(dBFS): -12 dB; Phase(degrees): 90.000
Tone 2: Frequency(MHz): 10.000933; Scale(dBFS): -6 dB; Phase(degrees): 90.000
Channel Q
Tone 1: Frequency(MHz): 5.000232
Tone 2: Frequency(MHz): 10.000933; Scale(dBFS): -9 dB; Phase(degrees): 45.000

FPGA Settings
Transmit / DDS
cf-ad9361-dds-core-lpc
TX 1 DDS Mode: DAC Buffer Output
TX 2 DDS Mode: DAC Buffer Output
DAC Buffer Settings
File Selection: sinewave_0.3.mat — Load
Waveform loaded successfully.
DAC Channels
- [x] voltage0
- [x] voltage1
- [ ] voltage2
- [ ] voltage3

High Level Device Control



ADI IIO Oscilloscope
File Settings Help
DMM | Debug | AD9081
Refresh

RX Chain
Global RX
Test Mode: off
Nyquist Zone: odd
Loopback Mode: 0
PFIR Config: (None)
ADC Frequency: 6000,00
Channel1 [FDDC0->CDDC0->ADC0]
NCO Frequency: 0,000000 | 0,000
Main NCO Frequency: 1000,000000
Channel2 [FDDC1->CDDC1->ADC1]
NCO Frequency: 0,000000
Main NCO Frequency: 1000,000000

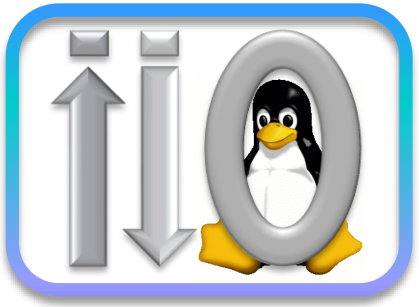
TX Chain
Global TX
DAC Frequency: 12000,00
Channel1 [FDUC0->CDUC0->DAC0]
[x] Enable
Channel
NCO Frequency: 0,000000 | 0,000 | [ ] Test Tone Enable
Gain Scale: 0,5001221 | Test Tone Scale: 0,0000000
Main
Main NCO Frequency: 1000,000000 | 0,000

DDS
axi-ad9081-tx-hpc
TX 1
DDS Mode: One CW Tone
Single Tone
Tone 1
Frequency(MHz): 39,986266
Scale(dBFS): -12 dB
Phase(degrees): 90,000
TX 2
DDS Mod…
Single To…
Tone 1
Frequency…
Scale(dBF…
Phase(deg…

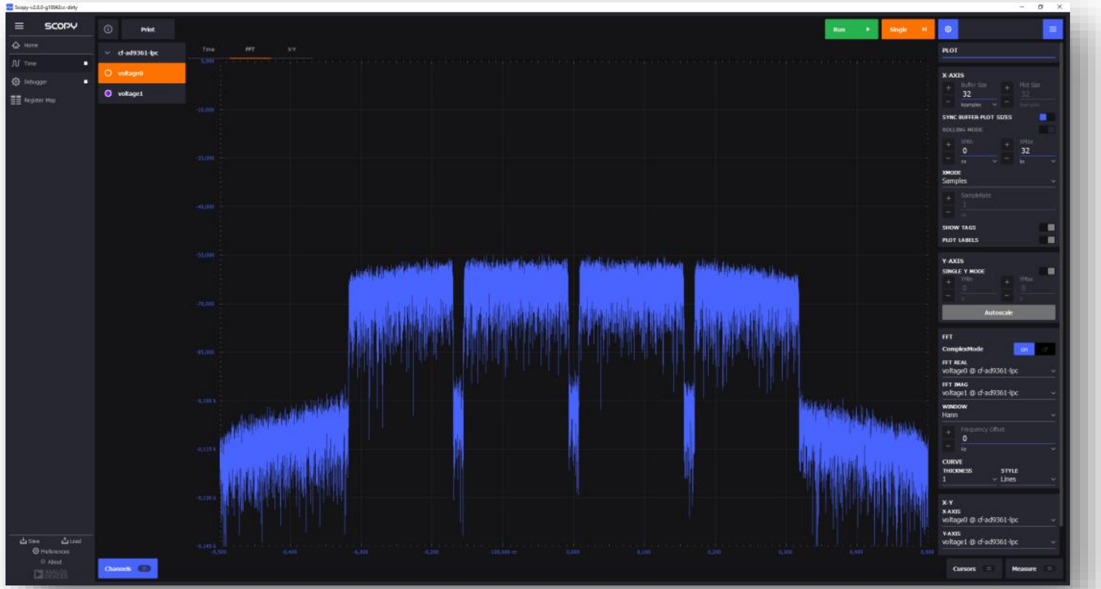
ADI IIO Oscilloscope
File Settings Help
ADRV9002 | DMM | Debug
Controls | Block Diagram
ADRV9002 Global Settings
Load Profile: (None)
Temperature (°C): 54,0000
ADRV9002 Receive Chain
RX 1
Hardware Gain (dB): 36,00 | Digital Gain Control: Gain_Correction_manual_control
Gain Control: spi | Interface Gain(dB): 0dB
ENSM: rf_enabled | Port Enable: spi
Powerdown: [ ] Enable | Dynamic Adc Switch: [ ] Enable
NCO (Hz): 0,000000 | Local Oscillator (MHz): 2400,000000
Decimated Power (dB): 4,2500 | RSSI (dB): 4,1890
Bandwidth (MHz): 10,0000 | Sampling Rate (MSPS): 15,3600
Tracking: [x] Quadrature FIC [x] AGC [x] RFDC [x] BBDC Rejection [ ] Quadrature Poly [ ] HD2 [x] RSSI
RX 2
Hardware Gain (dB): 36,00
Gain Control: spi
ENSM: rf_enabled
Powerdown: [ ] Enable
NCO (Hz): 0,000000
Decimated Power (dB): 60,7500
Bandwidth (MHz): 10,0000
Tracking: [x] Quadratu… [x] BBDC Re… [ ] HD2
ADRV9002 Transmit Chain
FPGA Settings
Reload Settings

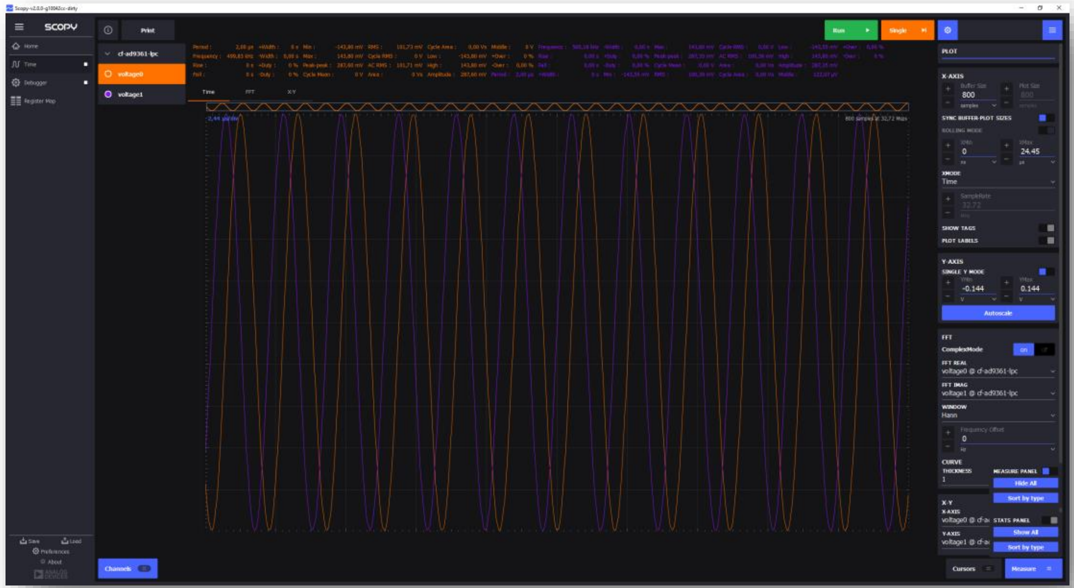
IIO – libiio – Command line tools

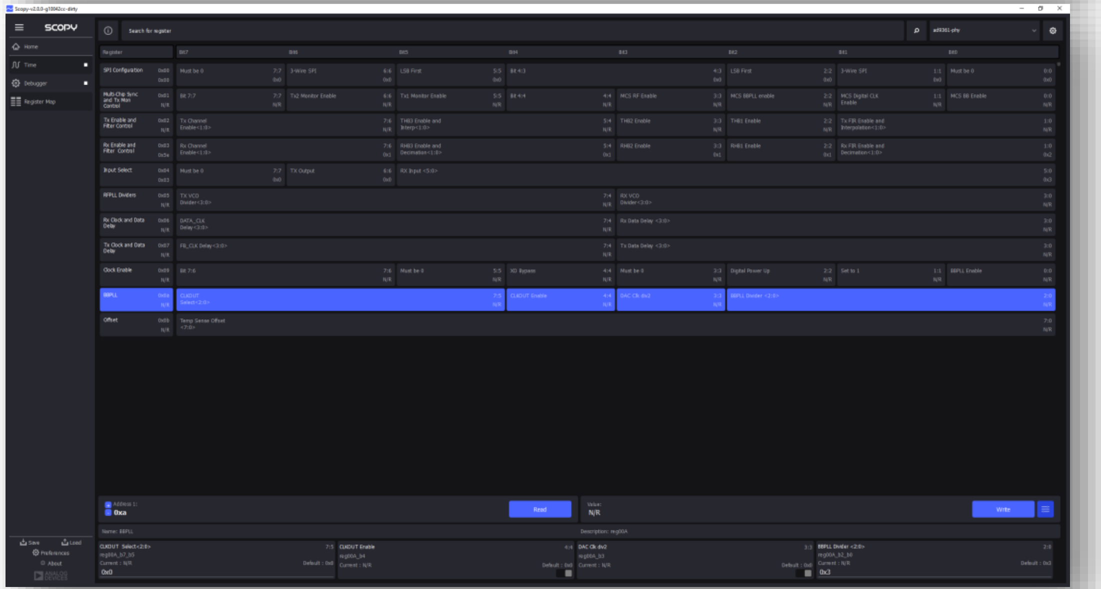
- **iio_info** : Information about all IIO devices, backends and context attributes
  - `iio_info -s`
  - `iio_info –u ip:192.168.2.1`
- **iio_attr** : Read and write IIO attributes
  - `iio_attr -c ad9361-phy altvoltage0 frequency 2450000000`
- **iio_readdev** : Read samples from an IIO device
  - `iio_readdev -u usb:1.100.5 -b 100000 cf-ad9361-lpc | pv > /dev/null`
- **iio_writedev** : Write samples to an IIO device
  - `iio_readdev -b 100000 cf-ad9361-lpc | iio_writedev -b 100000 cf-ad9361-dds-core-lpc`
- **iio_reg** : Read or write SPI, I2C and MMIO registers in an IIO device (useful to debug drivers)
  - `iio_reg adrv9009-phy 0`



What's next?





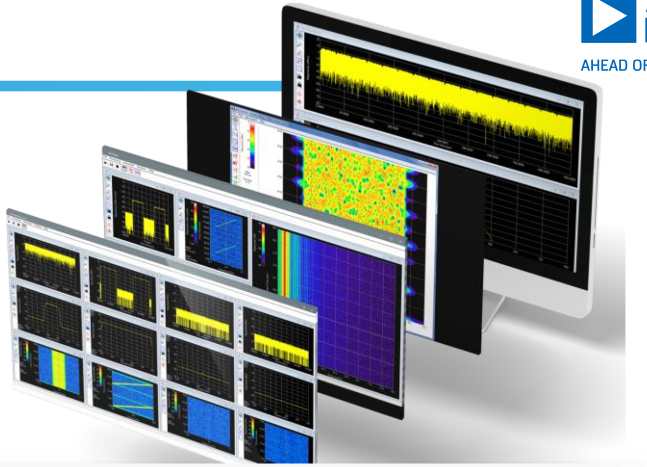


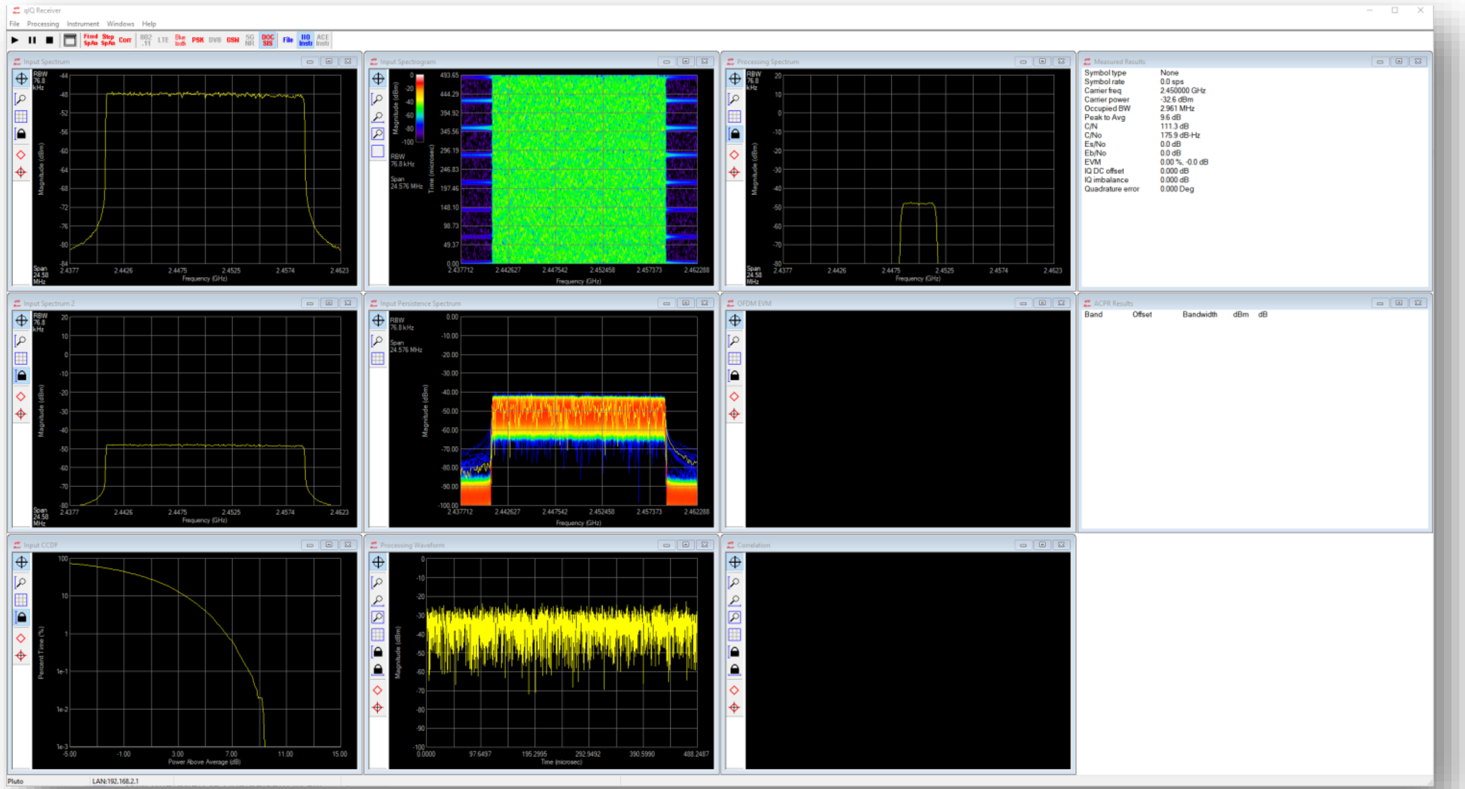
- **Scopy as generic IIO instrument**
  - **Will replace OSC in future**

qIQ Receiver

- qIQ Receiver
  - 3rd party tool
  - Windows application which provides receive capability for evaluating signal quality for transceiver and direct sampling chips from ADI
  - qIQ Receiver is perfect for:
    - Evaluating the chip's performance using an external signal source
  - Supports basically any TRX, MxFE, ADC with IIO driver support

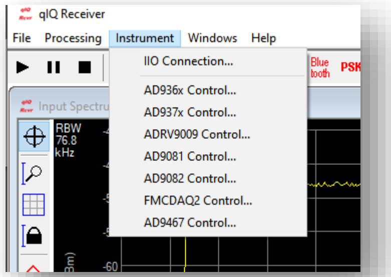
QIQ Systems





Measured Results

Symbol type	None
Symbol rate	0.0 sps
Carrier freq	2.450000 GHz
Carrier power	-32.6 dBm
Occupied BW	2.961 MHz
Peak to Avg	9.6 dB
C/N	111.3 dB
C/No	175.9 dB-Hz
Es/No	0.0 dB
Eb/No	0.0 dB
EVM	0.00 %, -0.0 dB
IQ DC offset	0.000 dB
IQ imbalance	0.000 dB
Quadrature error	0.000 Deg

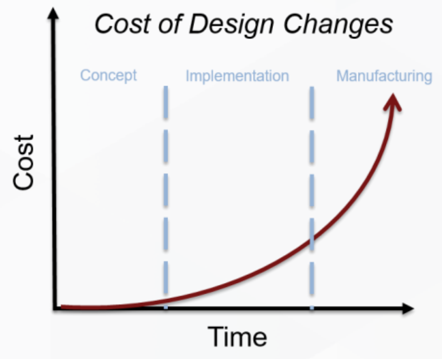


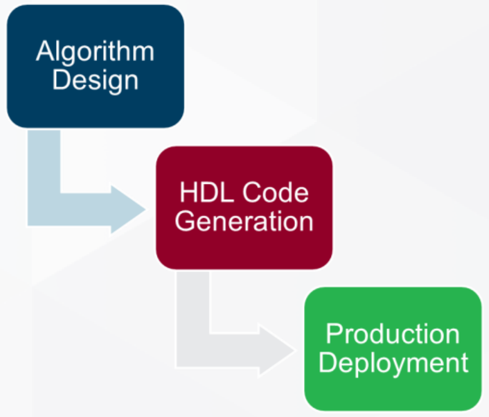
*"I want to **model and process data** using familiar tools."*
- Modelling/Domain Expert

MATLAB Simulink

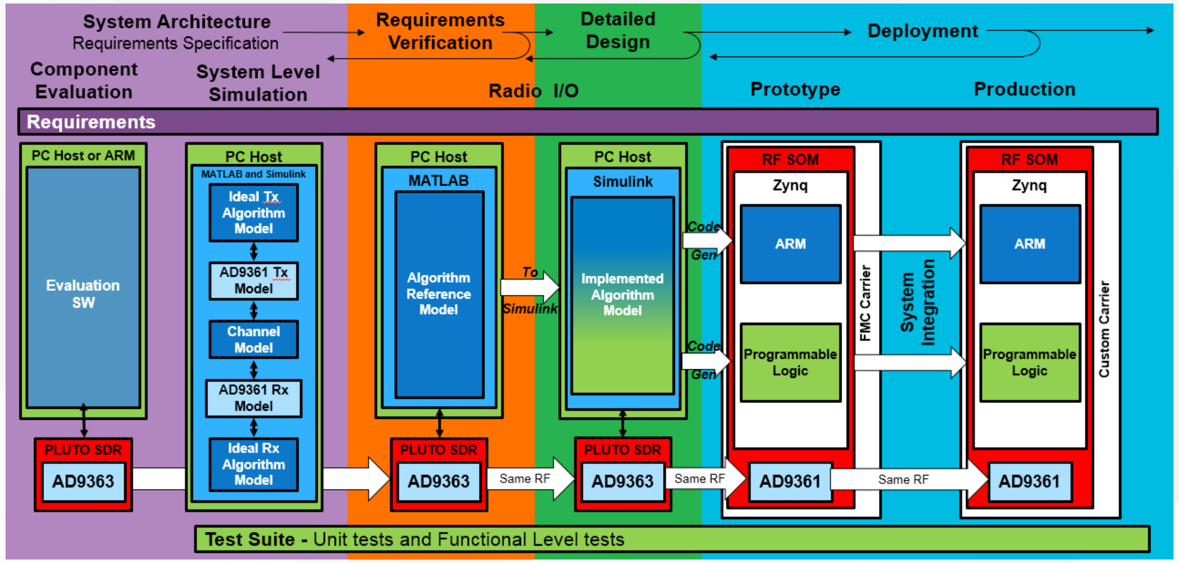
MATLAB Simulink Integration

- Industry standard tools for streamlined workflow and development / integration process
  - From concept to production
- Single environment for
  - Hardware evaluation
  - System simulation
  - Algorithm development and validation
  - Requirements verification
  - Data streaming
  - HDL & SW Code generation
- Model-Based Design
  - Modeling and simulation of the RF signal chain
  - Development, modeling, and simulation of communications algorithms
    - MATLAB and Simulink
    - GnuRadio
  - Testing and verification of algorithms with real-world data
    - Streaming from RF hardware (hardware in the loop)
  - Deployment of communications system to hardware for prototyping and production
    - Code generation and targeting





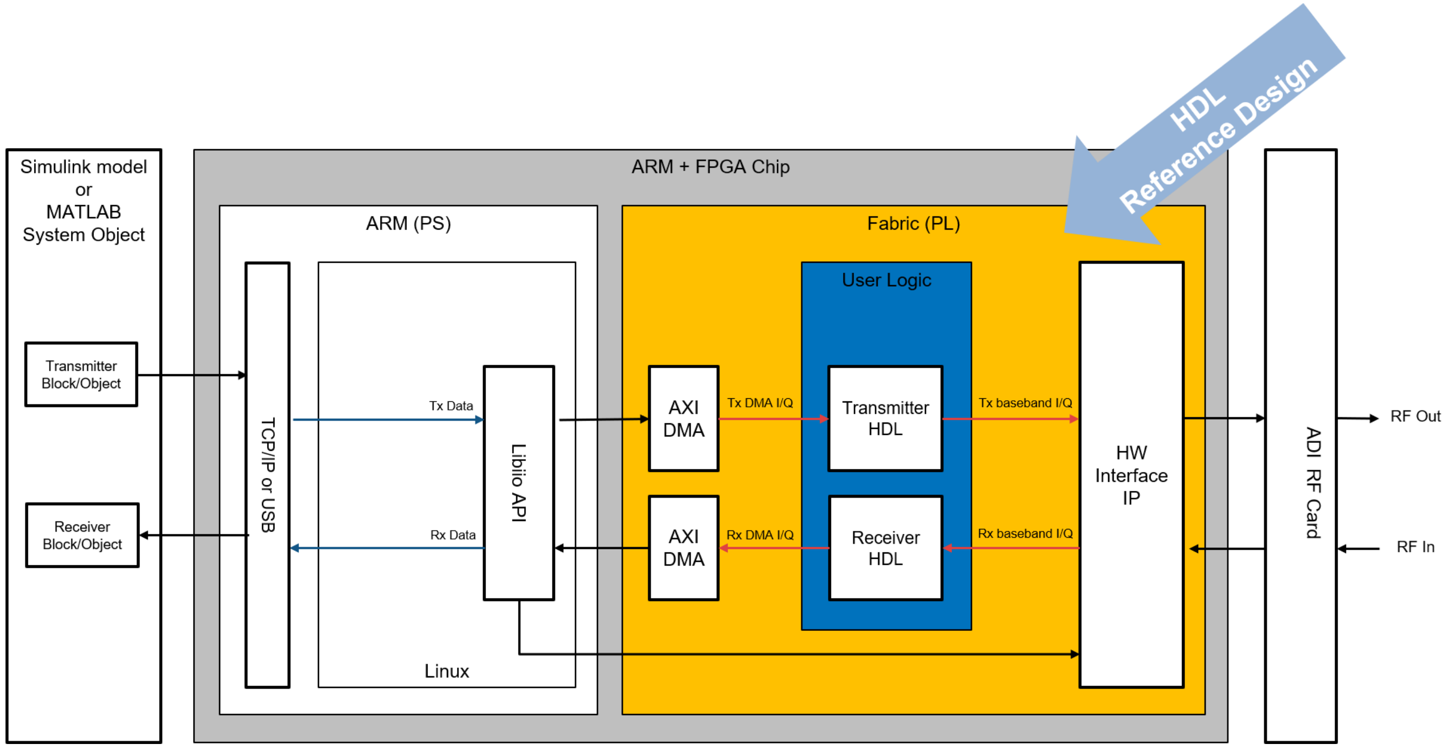
MATLAB Simulink



MATLAB: Advanced Application Level Support

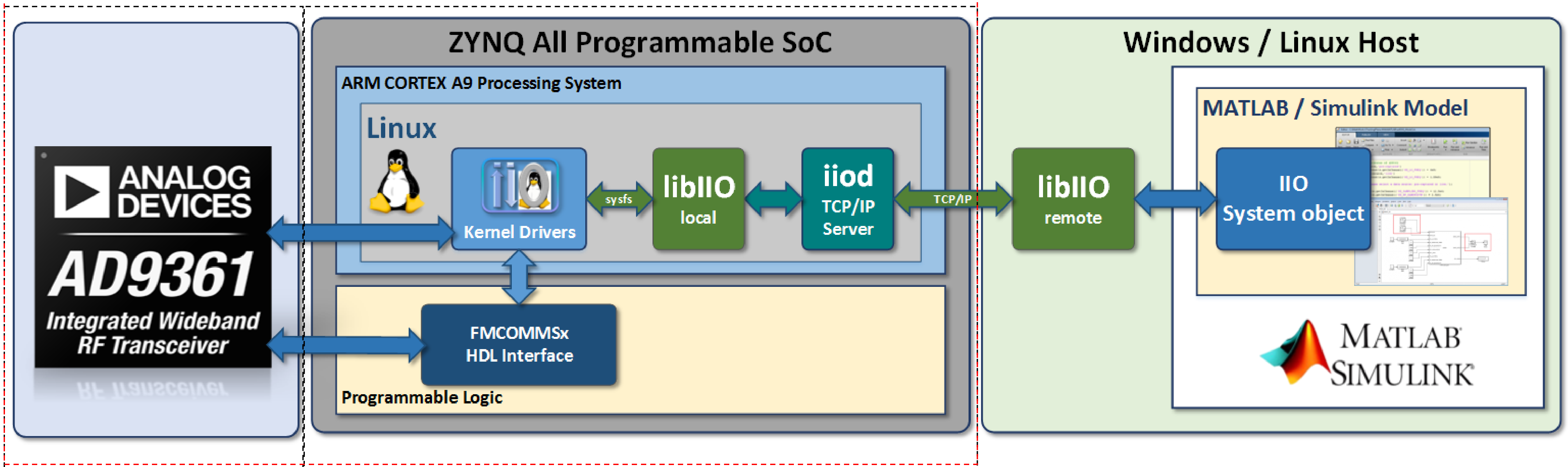
MATLAB

- All components rely on libiio or tinyiiod
- MATLAB data streaming and HDL/C/C++ targeting
  - ADI-TRX TOOLBOX
    ADI TRANSCEIVER TOOLBOX FOR MATLAB & SIMULINK
  - ADI-HSX TOOLBOX
    ADI HIGH SPEED CONVERTER TOOLBOX FOR MATLAB & SIMULINK
  - ADI-SENSOR TOOLBOX
    ADI ACCELEROMETER & GYROSCOPE TOOLBOX FOR MATLAB & SIMULINK
  - ADI-TOF TOOLBOX
    ADI TIME OF FLIGHT TOOLBOX FOR MATLAB & SIMULINK



IIO Driver

ADALM-PLUTO,
ADRV9361-Z7035 RF SOM, ADRV9364-Z7020 RF SOM, ADRV9009-ZU11EG RF SOM, ...



FMCOMMS2, FMCOMMS3,
FMCOMMS4, FMCOMMS5,
ADRV9009 AD9144
AD9680 AD9172 AD9213
AD9081 AD9152 ...

Xilinx Zed Board, Xilinx ZC702, Xilinx ZC706,
Xilinx KC705, Xilinx KCU105,
Xilinx ZCU102, Xilinx VCU118, Xilinx VCU128 Xilinx VMK180, Xilinx VCK190
Intel Arria 10 SoC, Intel Stratix 10 SoC

ADALM-PLUTO
Aka PlutoSDR

ADALM-PLUTO (PlutoSDR)

Platform to get familiar with ADI's SDR infrastructure, focus on streaming

- Processor
  - Dual core Arm® Cortex®-A9 (667 MHz each)
  - L1 cache: 32 kB instruction, 32 kB data
  - L2 cache: 512 kB
- FPGA
  - Artix®-7 fabric
  - 28k logic cells
  - 2.1 Mb block RAM
  - 80 DSP slices

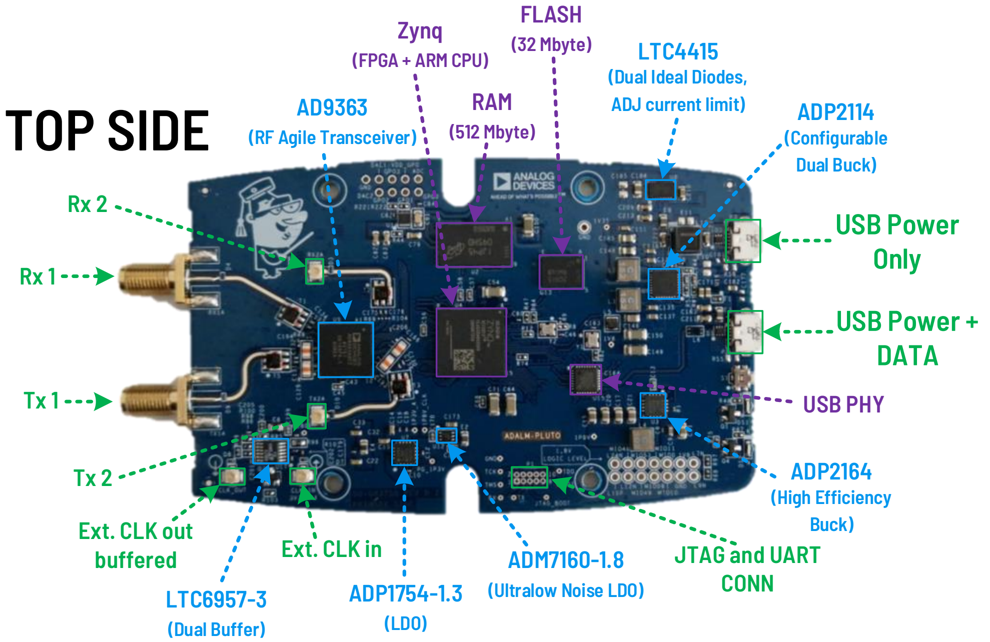
Xilinx® XCZ7010

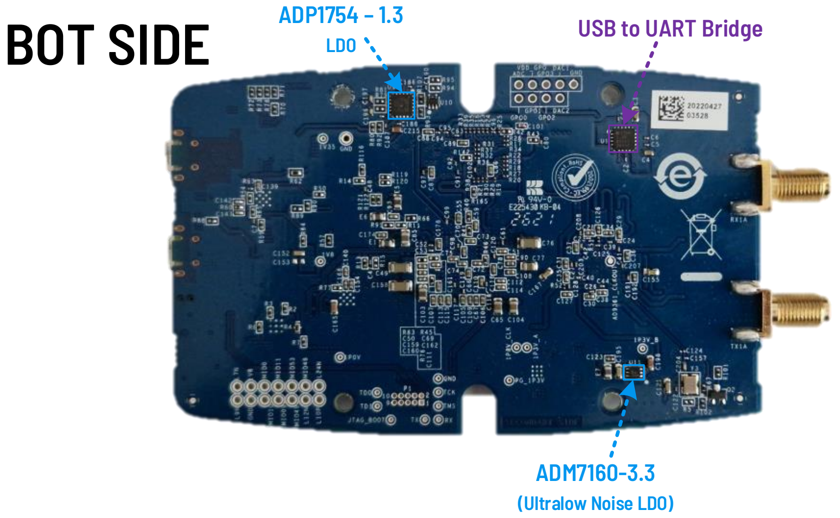
- 512 MB DDR3L
- 32 MB SPI Flash

- Radio
  - Up to 20 MHz RF bandwidth
  - 128-tap FIR filters for equalization
  - 325 MHz to 3800 MHz tuning range
  - 1 Rx, 1 Tx, half or full duplex

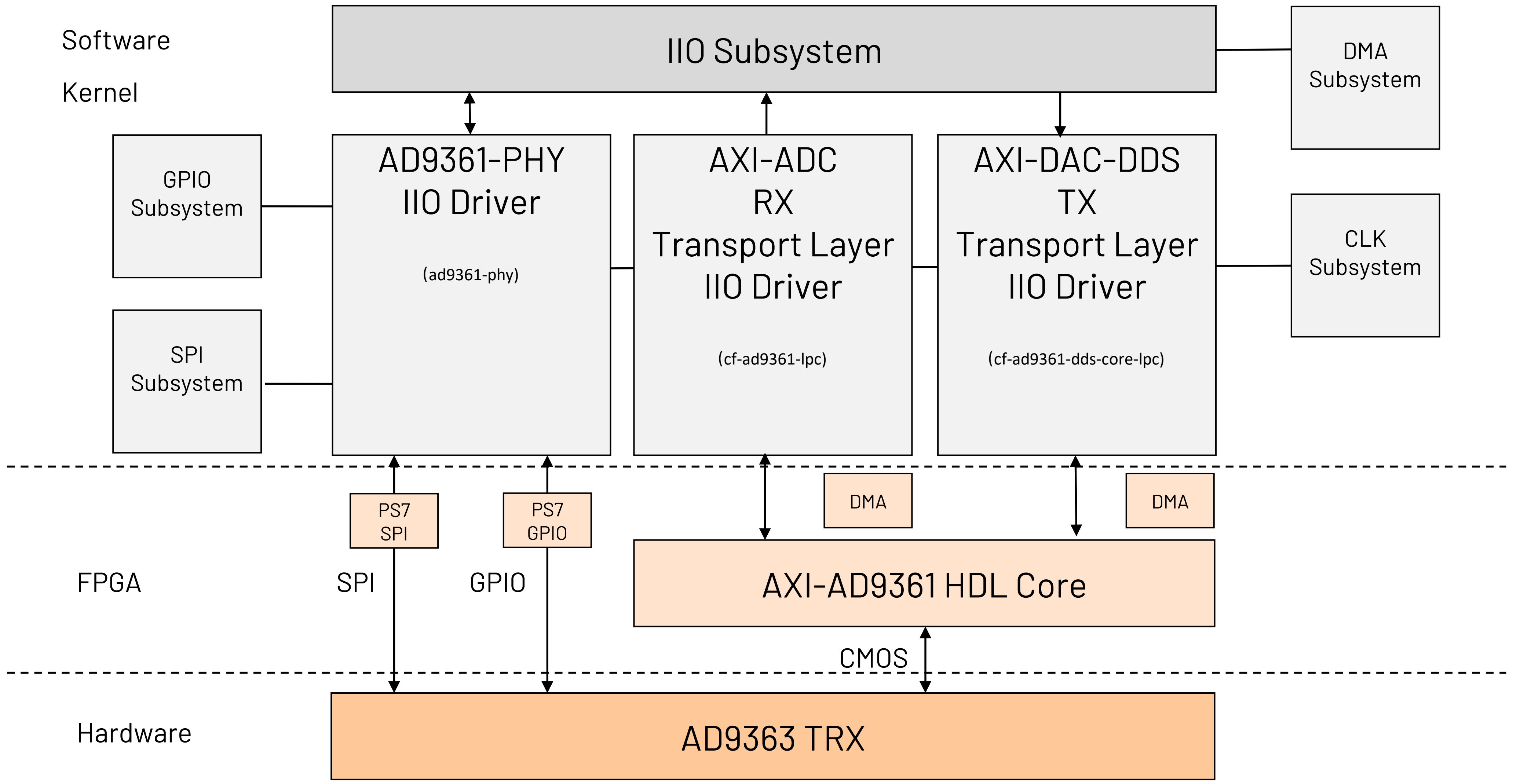
ADI AD9363

- 40 MHz TCVCXO ref clock with ±1 ppm stability
- USB 2.0 (OTG controller + PHY)
- Full Linux®-based reference design
- Fully integrated and tested system
- Weight: 114 g
- Size: 117 mm × 79 mm × 24 mm
- Component temperature rating: 10°C to 40°C
- Typical power consumption under 2 W
- SMA connectors: 2





Software, Programmable Logic & Hardware

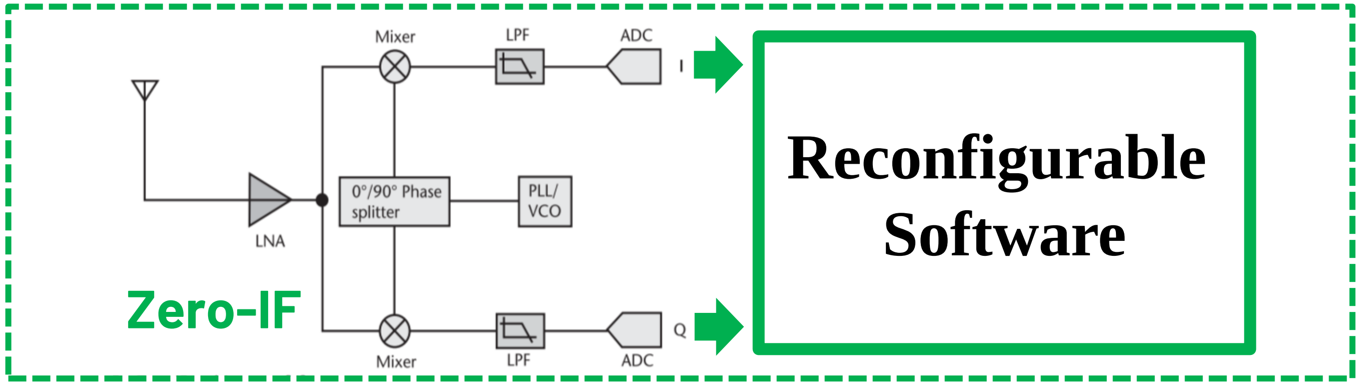


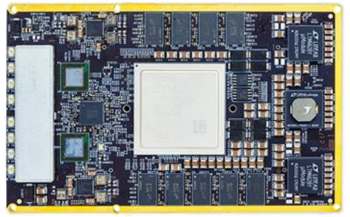
Hands on section

Workshop Contents

1. What is SDR (Software Defined Radio)?

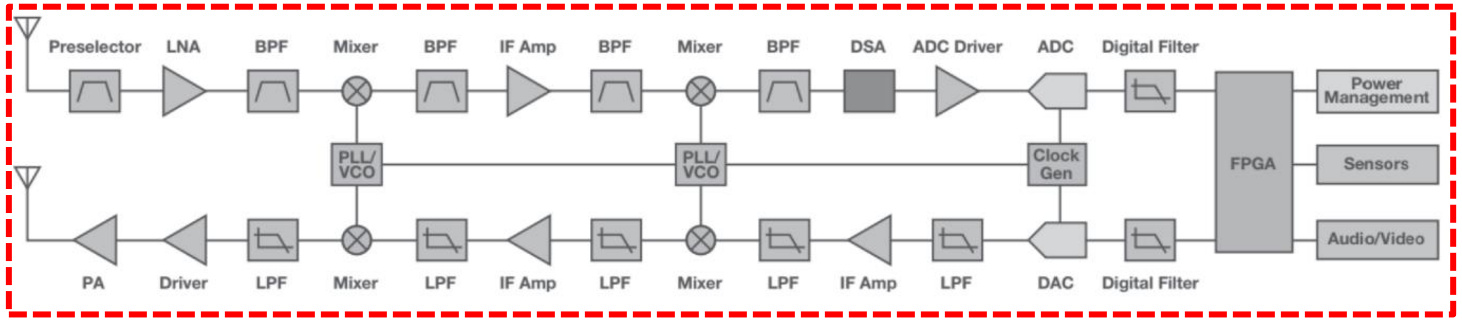
Reconfigurable, multipurpose radio:

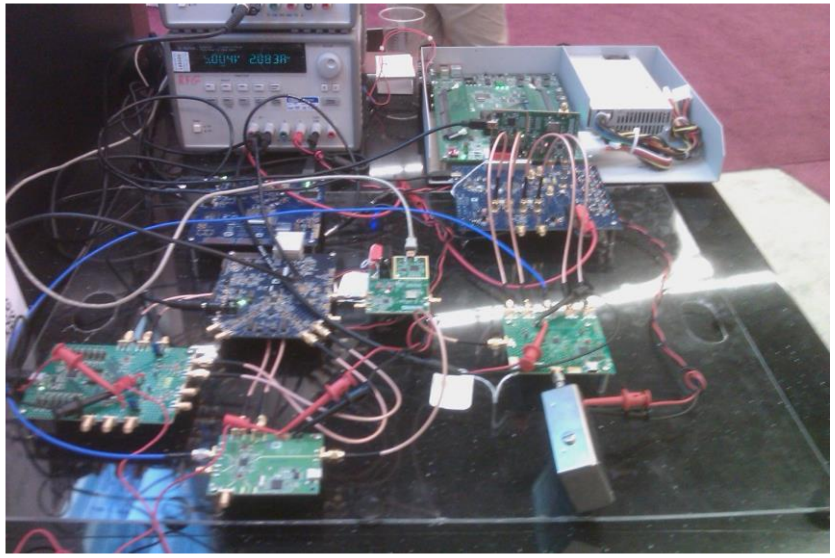




(Not SDR)

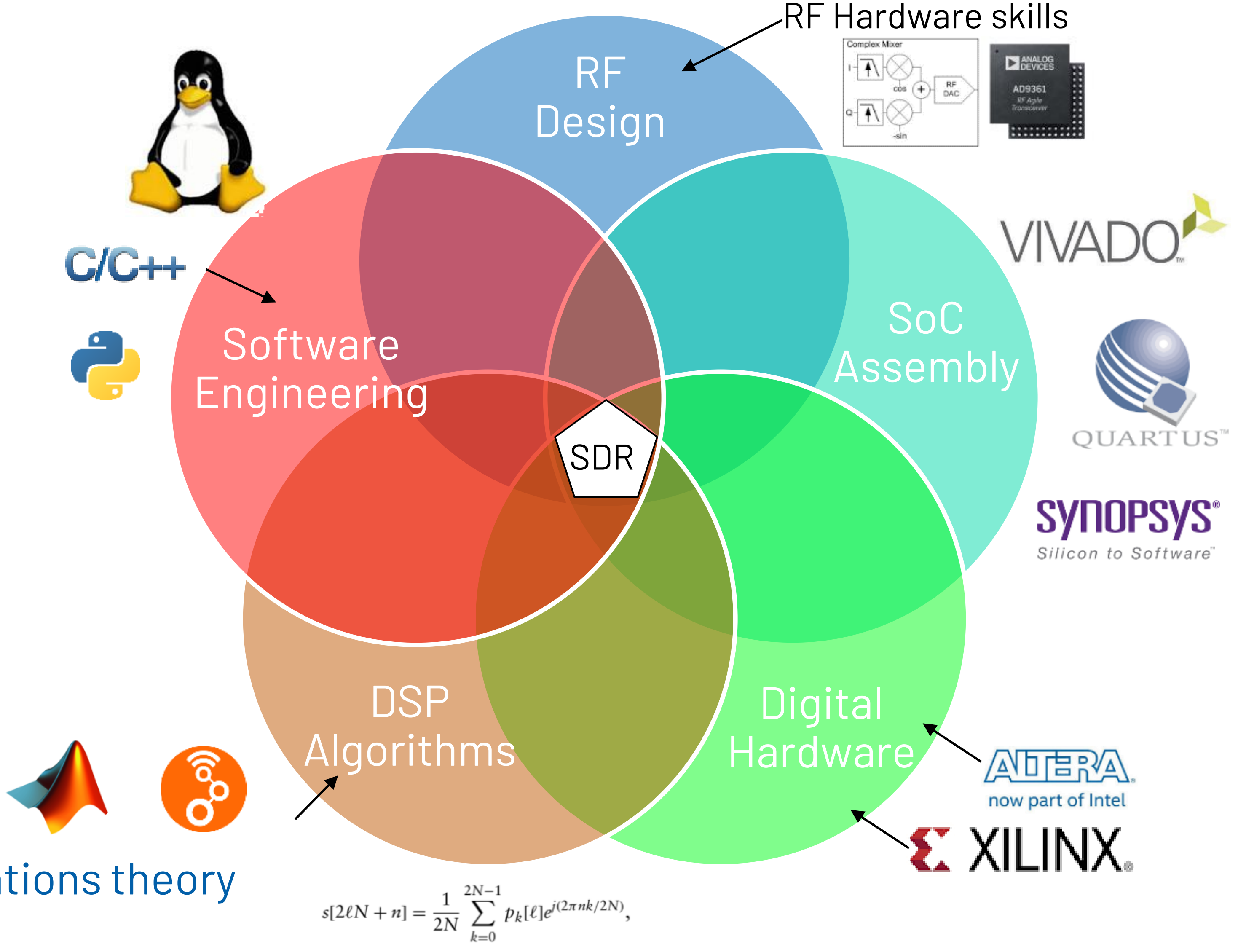
Optimized only for a few applications (Super Heterodyne):





1. 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



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

Hardware

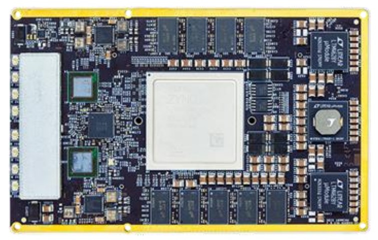
ADALM Pluto

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

Jupiter

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

Talise SOM



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

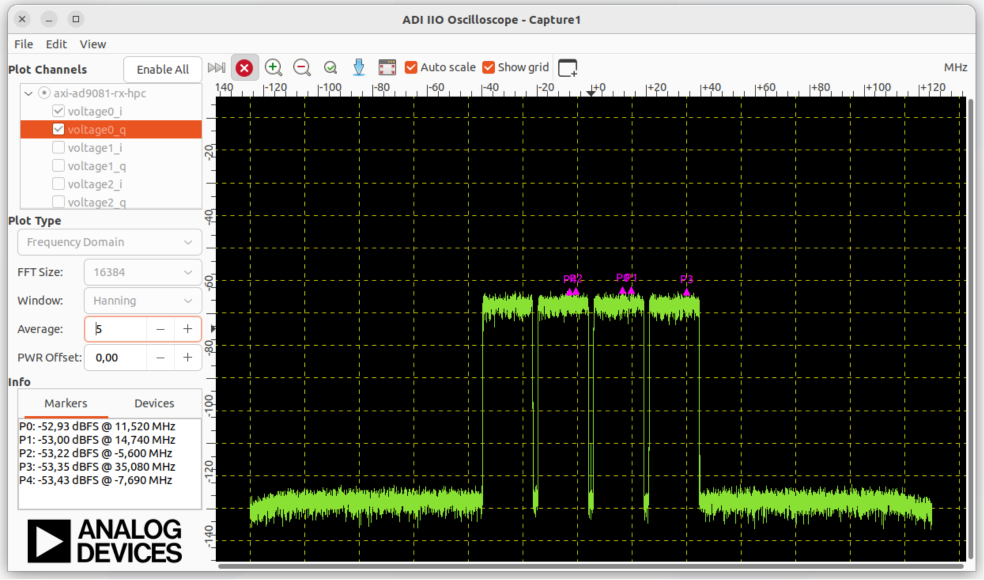
2. IIO-Scope

Analog Devices Inc. Industrial I/O Oscilloscope

HEAD-g938d5a0

Homepage

- Capture and display data
  - Time domain (with trigger support)
  - Frequency domain
  - Constellation plot
  - Markers
  - Math operations



Plot Type: Frequency Domain

FFT Size: 16384

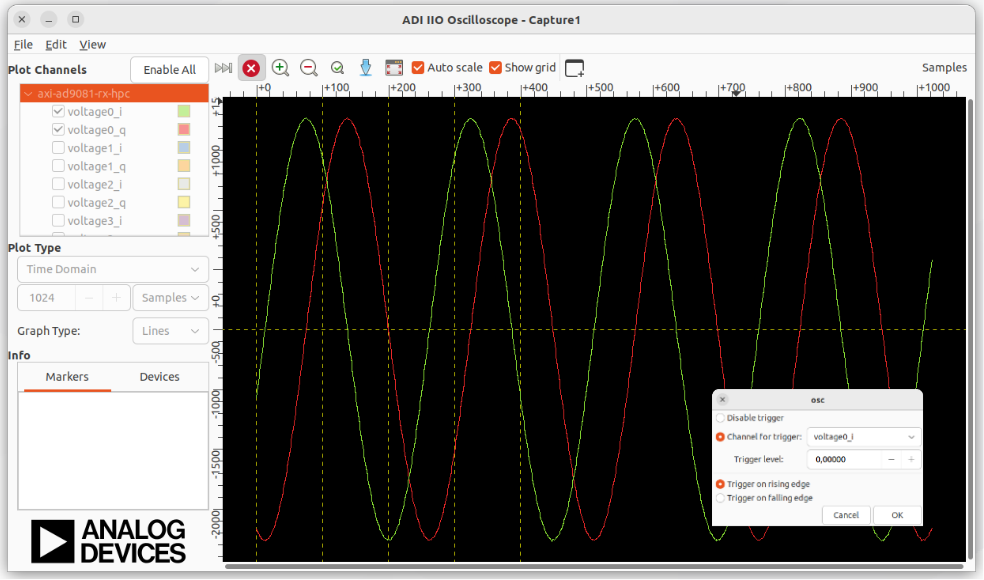
Window: Hanning

Average: 5

PWR Offset: 0,00

Info

```
P0: -52,93 dBFS @ 11,520 MHz
P1: -53,00 dBFS @ 14,740 MHz
P2: -53,22 dBFS @ -5,600 MHz
P3: -53,35 dBFS @ 35,080 MHz
P4: -53,43 dBFS @ -7,690 MHz
```



Plot Type: Time Domain

1024 Samples

Graph Type: Lines

Disable trigger

[✓ Channel for trigger:] voltage0_i

Trigger level: 0,00000

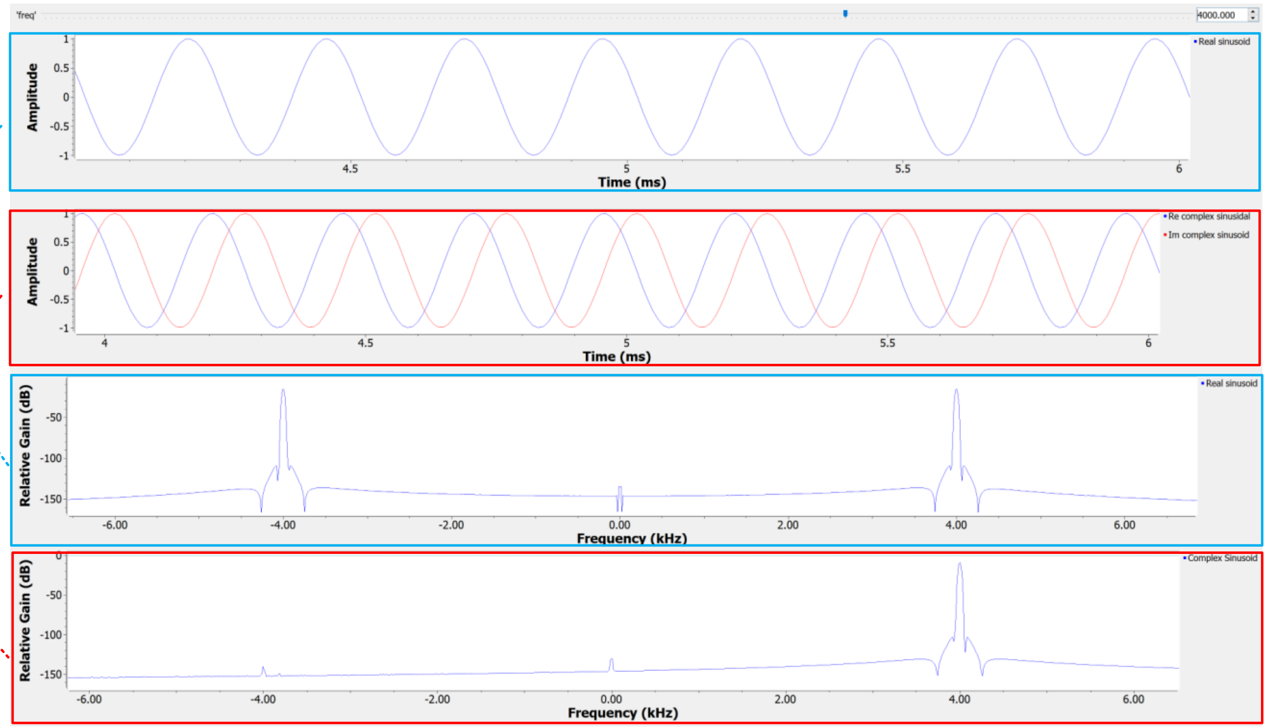
[✓ Trigger on rising edge]

Trigger on falling edge

3. Transmit and receive a complex sinusoid with Python

What is a complex sinusoid?

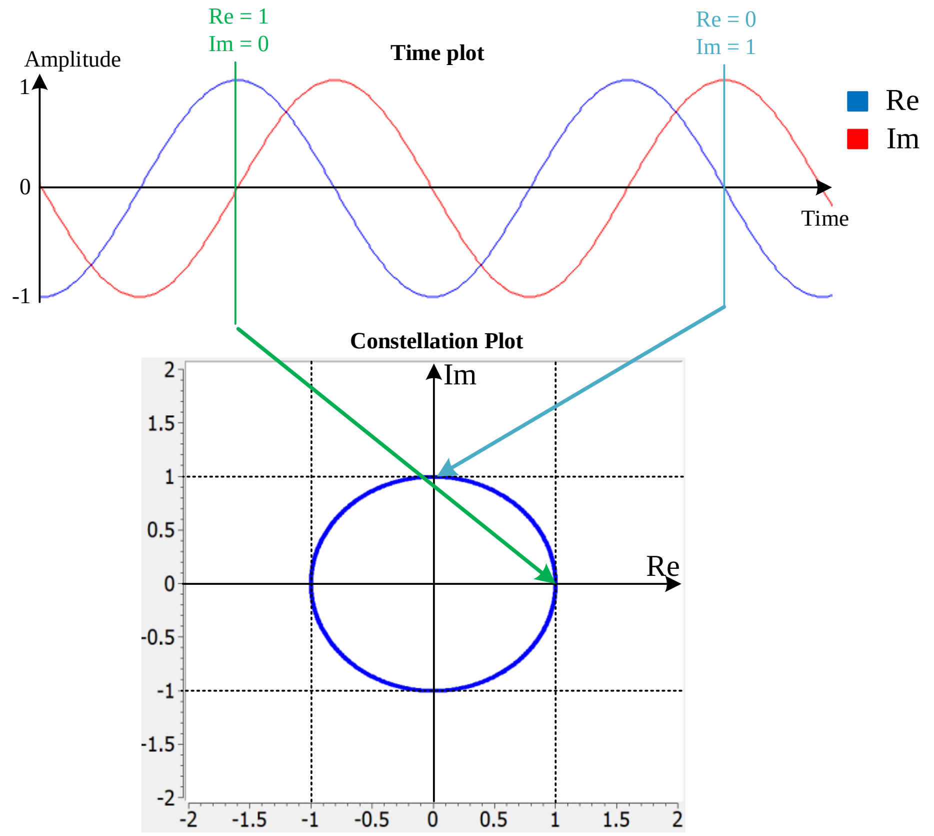
- Real sinusoid: $e^{j2\pi ft} + e^{-j2\pi ft} = 2\cos(2\pi ft)$
- Complex sinusoid: $e^{j2\pi ft} = \underbrace{\cos(2\pi ft)}_{\text{I (In phase)}} + j\,\underbrace{\sin(2\pi ft)}_{\text{Q (Quadrature phase)}}$



Real Sinusoid

Complex Sinusoid

- Complex sinusoid (Constellation Plot):



3.Transmit and receive a complex sinusoid with Python

What is a PyADI-IIO?

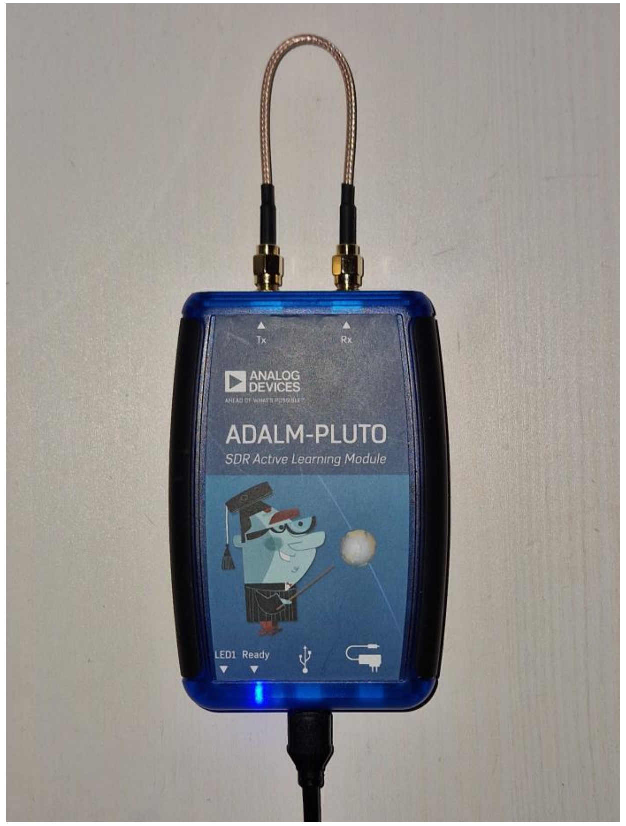
ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™

PyADI-IIO
PYTHON FOR ADI INDUSTRIAL I/O DEVICES

► **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

▶ **Connect Rx and Tx using the SMA cable from the kit (as in the picture from below) and connect the Pluto to the PC using the USB Cable provided.**



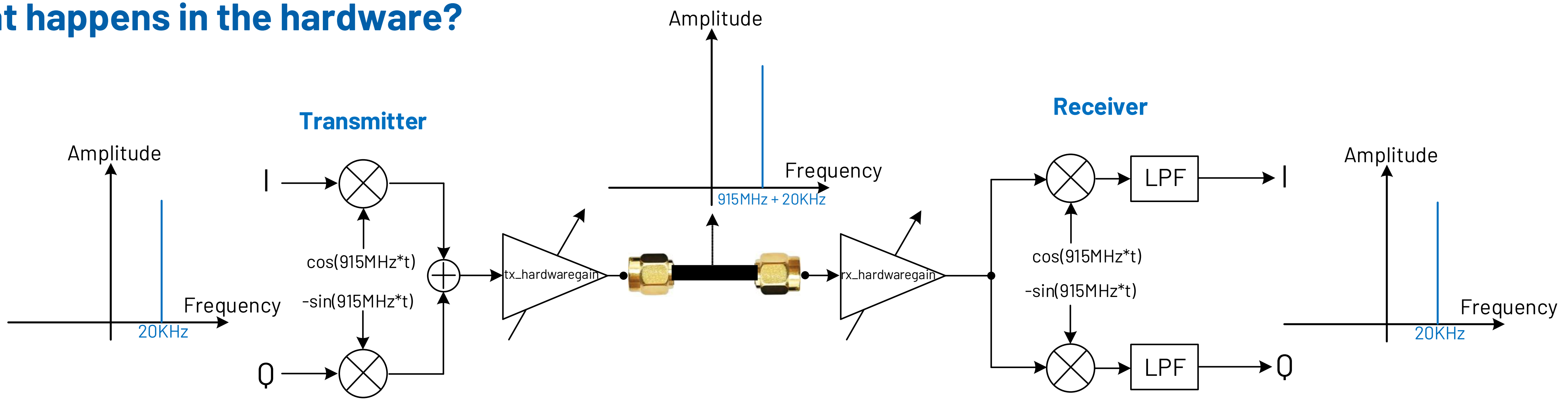
▶ **Go to** ~/Desktop/ftc23_sdr/python/**python_loopback_sine_pluto.py and open with Thonny Python IDE.**

3. Transmit and receive a complex sinusoid with Python

- **Open** ~/Desktop/ftc23_sdr/python/sinewave_loopback/**python_loopback_sine_pluto.py**:
- **What does this code do?**

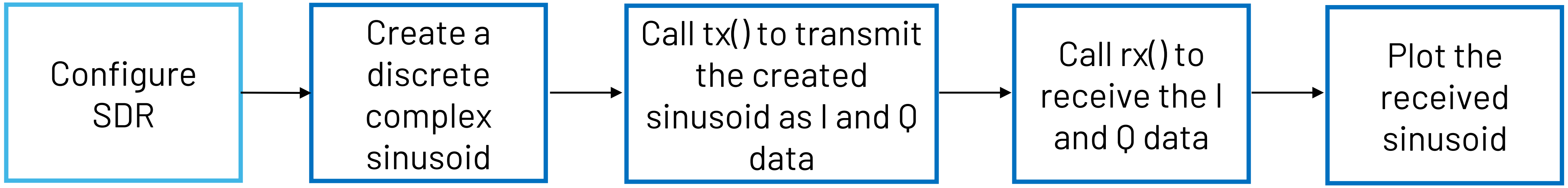
$$\begin{tikzcd}
\boxed{\text{Configure SDR}} \arrow[r] & \boxed{\text{Create a discrete complex sinusoid}} \arrow[r] & \boxed{\text{Call tx() to transmit the created sinusoid as I and Q data}} \arrow[r] & \boxed{\text{Call rx() to receive the I and Q data}} \arrow[r] & \boxed{\text{Plot the received sinusoid}}
\end{tikzcd}$$

- **What happens in the hardware?**

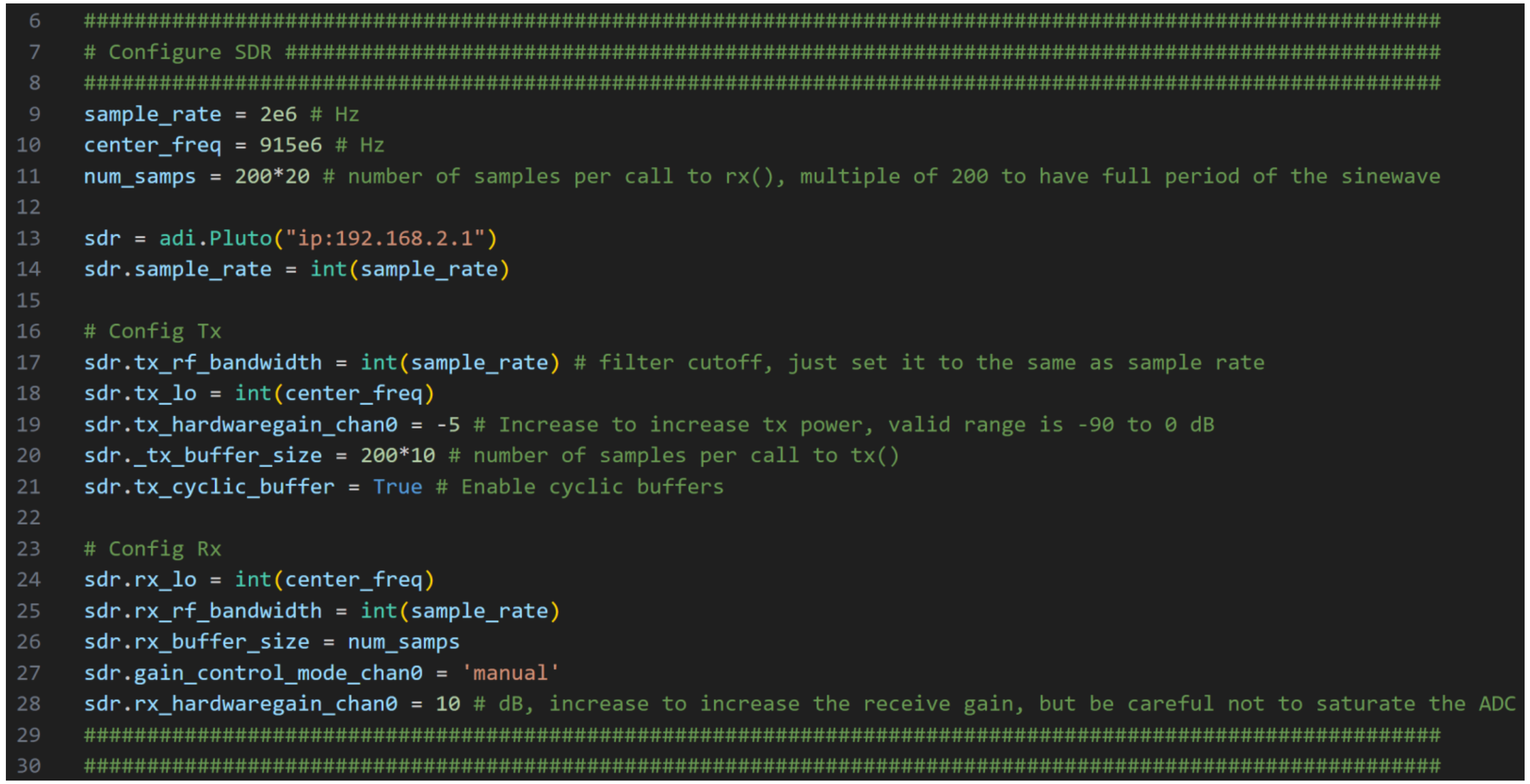


3.Transmit and receive a complex sinusoid with Python

▶ **Code snippets:**



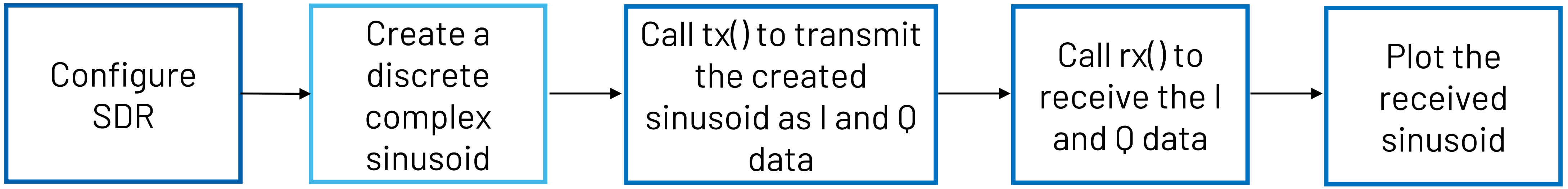
Configure SDR
Create a discrete complex sinusoid
Call tx() to transmit the created sinusoid as I and Q data
Call rx() to receive the I and Q data
Plot the received sinusoid

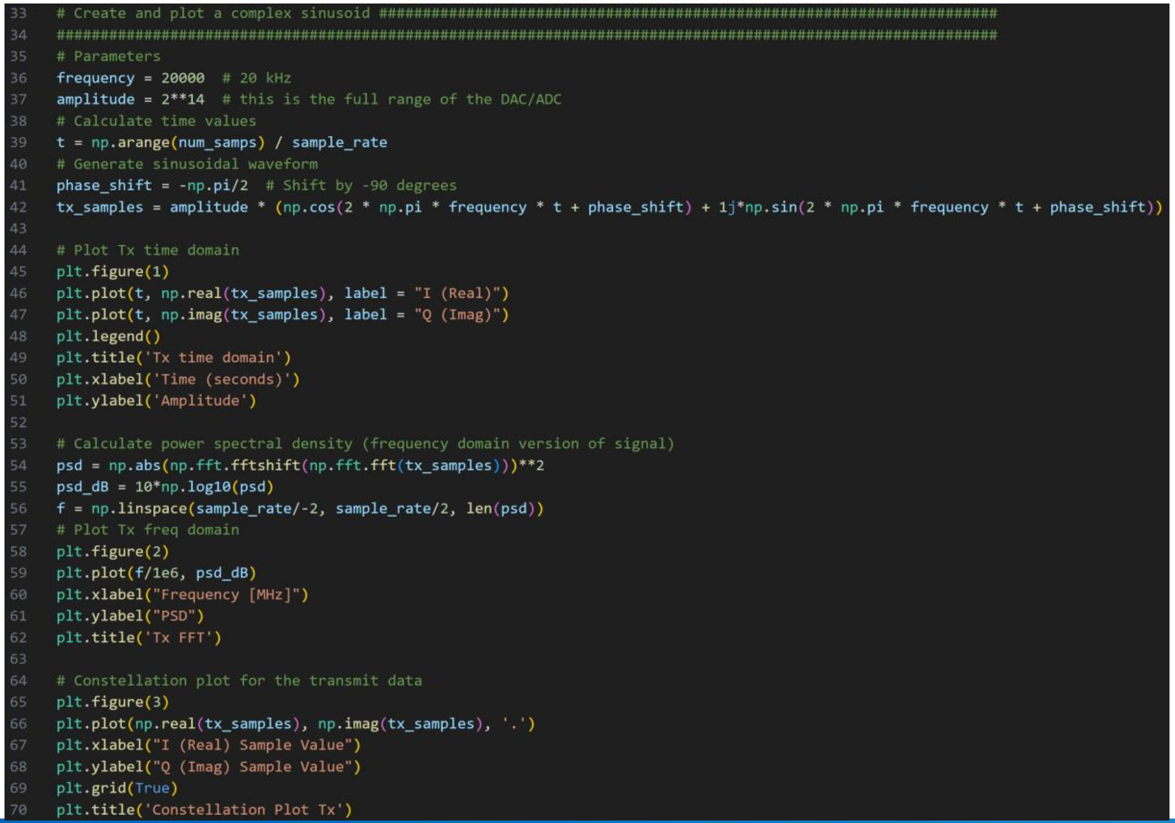


```
############################################################################################################
# Configure SDR ############################################################################################
############################################################################################################
sample_rate = 2e6 # Hz
center_freq = 915e6 # Hz
num_samps = 200*20 # number of samples per call to rx(), multiple of 200 to have full period of the sinewave

sdr = adi.Pluto("ip:192.168.2.1")
sdr.sample_rate = int(sample_rate)

# Config Tx
sdr.tx_rf_bandwidth = int(sample_rate) # filter cutoff, just set it to the same as sample rate
sdr.tx_lo = int(center_freq)
sdr.tx_hardwaregain_chan0 = -5 # Increase to increase tx power, valid range is -90 to 0 dB
sdr._tx_buffer_size = 200*10 # number of samples per call to tx()
sdr.tx_cyclic_buffer = True # Enable cyclic buffers

# Config Rx
sdr.rx_lo = int(center_freq)
sdr.rx_rf_bandwidth = int(sample_rate)
sdr.rx_buffer_size = num_samps
sdr.gain_control_mode_chan0 = 'manual'
sdr.rx_hardwaregain_chan0 = 10 # dB, increase to increase the receive gain, but be careful not to saturate the ADC
############################################################################################################
############################################################################################################
```

3. Transmit and receive a complex sinusoid with Python

▶ **Code snippets:**


Configure SDR
Create a discrete complex sinusoid
Call tx() to transmit the created sinusoid as I and Q data
Call rx() to receive the I and Q data
Plot the received sinusoid



```python
# Create and plot a complex sinusoid #######################################################################
############################################################################################################
# Parameters
frequency = 20000  # 20 kHz
amplitude = 2**14  # this is the full range of the DAC/ADC
# Calculate time values
t = np.arange(num_samps) / sample_rate
# Generate sinusoidal waveform
phase_shift = -np.pi/2  # Shift by -90 degrees
tx_samples = amplitude * (np.cos(2 * np.pi * frequency * t + phase_shift) + 1j*np.sin(2 * np.pi * frequency * t + phase_shift))

# Plot Tx time domain
plt.figure(1)
plt.plot(t, np.real(tx_samples), label = "I (Real)")
plt.plot(t, np.imag(tx_samples), label = "Q (Imag)")
plt.legend()
plt.title('Tx time domain')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')

# Calculate power spectral density (frequency domain version of signal)
psd = np.abs(np.fft.fftshift(np.fft.fft(tx_samples)))**2
psd_dB = 10*np.log10(psd)
f = np.linspace(sample_rate/-2, sample_rate/2, len(psd))
# Plot Tx freq domain
plt.figure(2)
plt.plot(f/1e6, psd_dB)
plt.xlabel("Frequency [MHz]")
plt.ylabel("PSD")
plt.title('Tx FFT')

# Constellation plot for the transmit data
plt.figure(3)
plt.plot(np.real(tx_samples), np.imag(tx_samples), '.')
plt.xlabel("I (Real) Sample Value")
plt.ylabel("Q (Imag) Sample Value")
plt.grid(True)
plt.title('Constellation Plot Tx')
```

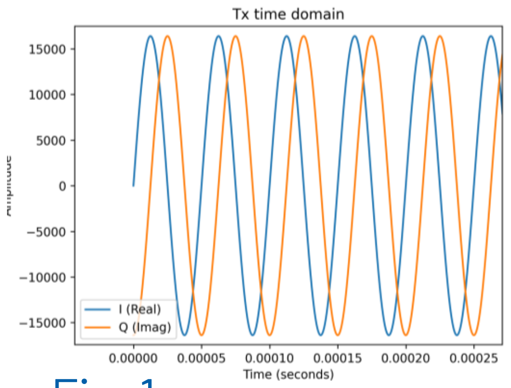


Fig. 1

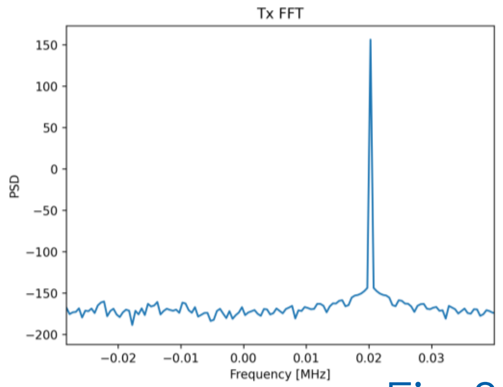


Fig. 2

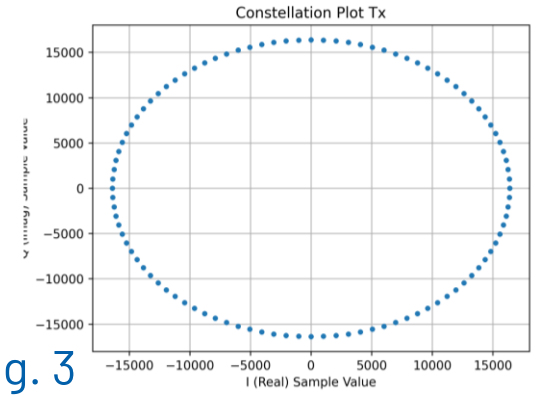
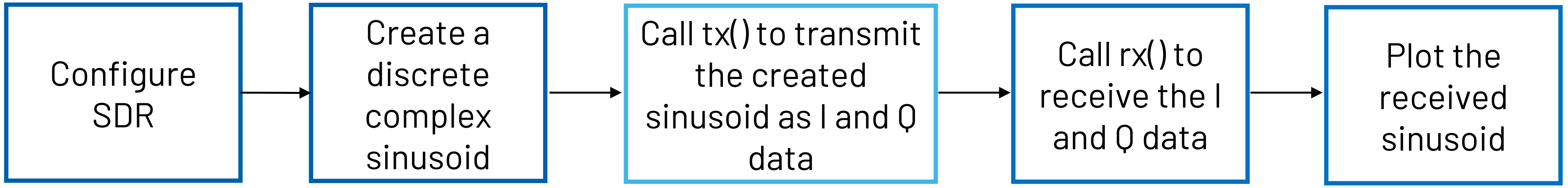


Fig. 3

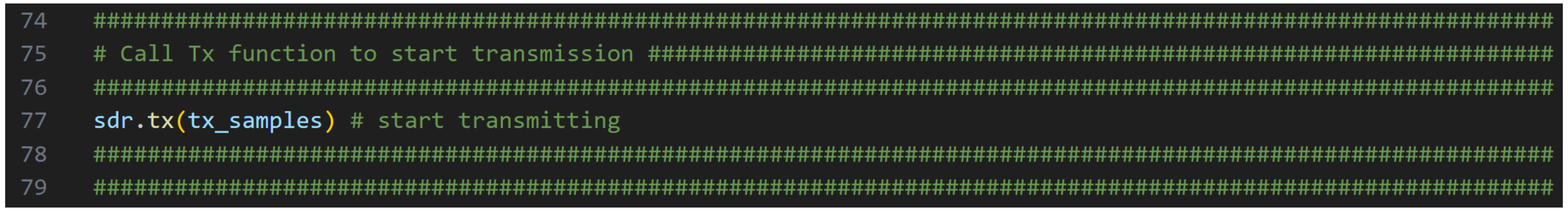
→ Constellation of a sinewave

3. Transmit and receive a complex sinusoid with Python

▶ **Code snippets:**



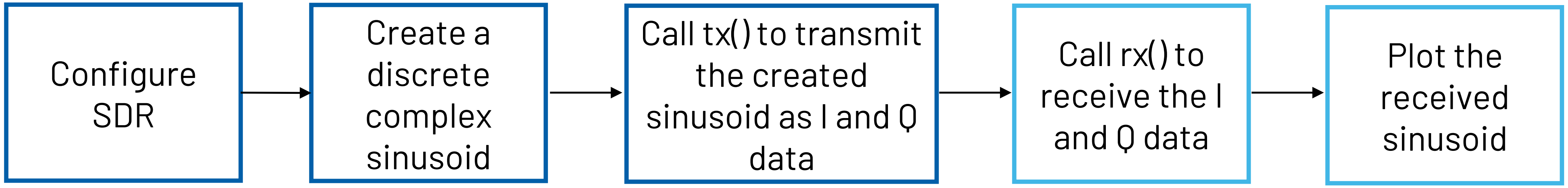
Configure SDR
Create a discrete complex sinusoid
Call tx() to transmit the created sinusoid as I and Q data
Call rx() to receive the I and Q data
Plot the received sinusoid

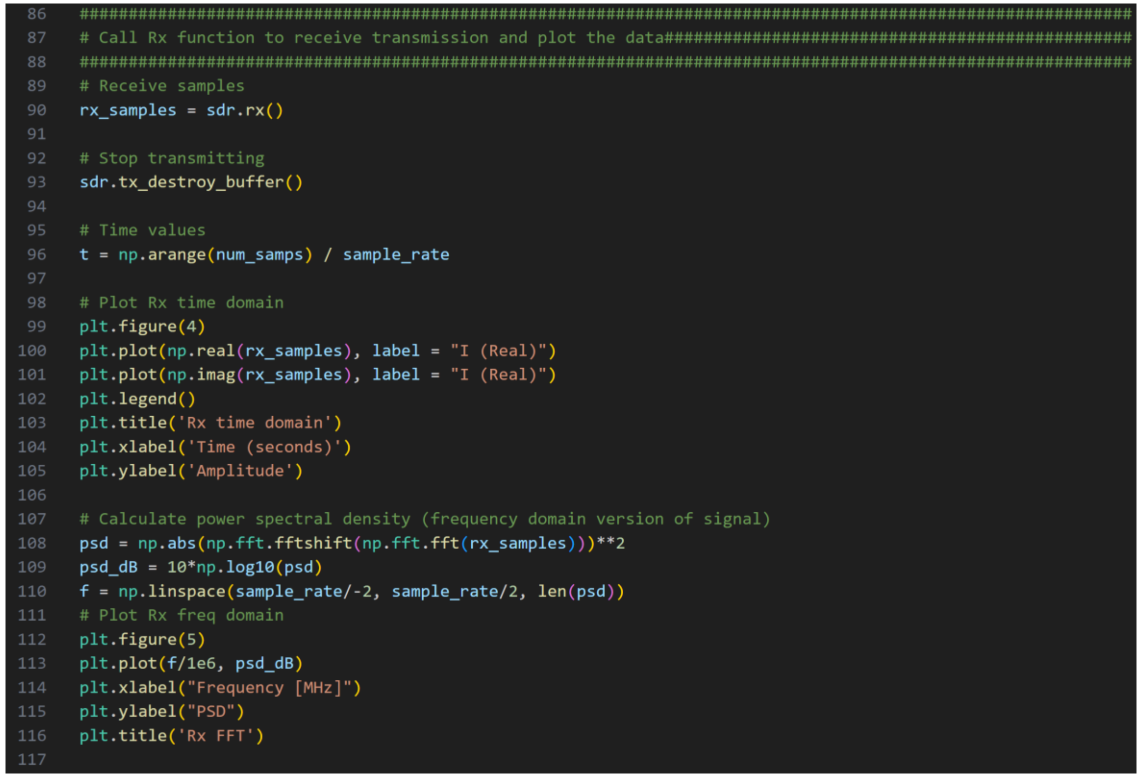


```
############################################################################################################
# Call Tx function to start transmission ###################################################################
############################################################################################################
sdr.tx(tx_samples) # start transmitting
############################################################################################################
############################################################################################################
```

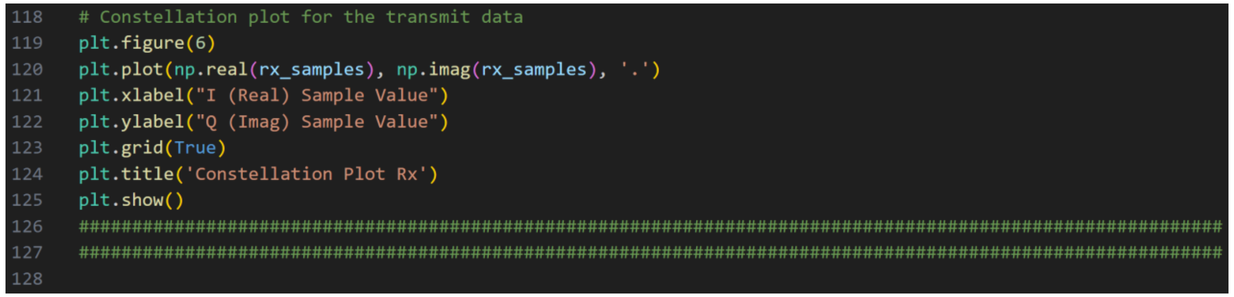
3. Transmit and receive a complex sinusoid with Python

▶ **Code snippets:**


Configure SDR
Create a discrete complex sinusoid
Call tx() to transmit the created sinusoid as I and Q data
Call rx() to receive the I and Q data
Plot the received sinusoid



```python
##########################################################################################
# Call Rx function to receive transmission and plot the data##############################
##########################################################################################
# Receive samples
rx_samples = sdr.rx()

# Stop transmitting
sdr.tx_destroy_buffer()

# Time values
t = np.arange(num_samps) / sample_rate

# Plot Rx time domain
plt.figure(4)
plt.plot(np.real(rx_samples), label = "I (Real)")
plt.plot(np.imag(rx_samples), label = "I (Real)")
plt.legend()
plt.title('Rx time domain')
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')

# Calculate power spectral density (frequency domain version of signal)
psd = np.abs(np.fft.fftshift(np.fft.fft(rx_samples)))**2
psd_dB = 10*np.log10(psd)
f = np.linspace(sample_rate/-2, sample_rate/2, len(psd))
# Plot Rx freq domain
plt.figure(5)
plt.plot(f/1e6, psd_dB)
plt.xlabel("Frequency [MHz]")
plt.ylabel("PSD")
plt.title('Rx FFT')

```



```python
# Constellation plot for the transmit data
plt.figure(6)
plt.plot(np.real(rx_samples), np.imag(rx_samples), '.')
plt.xlabel("I (Real) Sample Value")
plt.ylabel("Q (Imag) Sample Value")
plt.grid(True)
plt.title('Constellation Plot Rx')
plt.show()
##############################################################################################
##############################################################################################

```

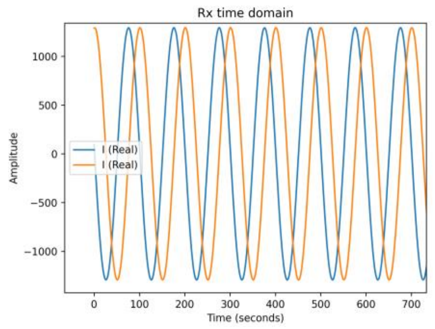


Fig. 4

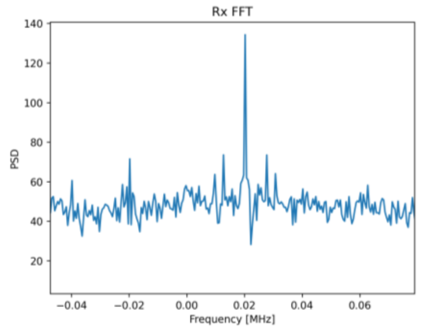


Fig. 5

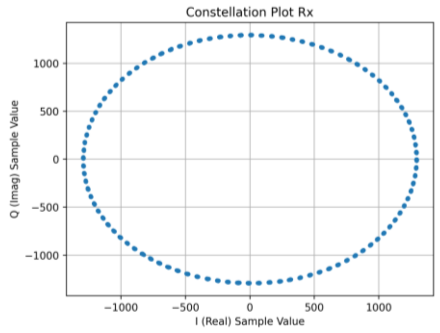


Fig. 6

3. Transmit and receive a complex sinusoid with Python

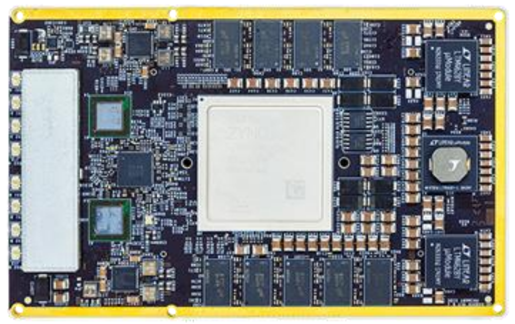
- **Using PyADI-IIO you can reuse the code and run it on more expensive SDRs like Jupiter and Talise.**

PyADI-IIO

PYTHON FOR ADI INDUSTRIAL I/O DEVICES

ADALM-PLUTO
- AD9363
- Zynq®-7010

Jupiter
- ADRV9002
- ZynqMP-ZU3EG



ADRV9009-ZU11EG
- 2× ADRV9009
- ZynqMP-ZU11EG

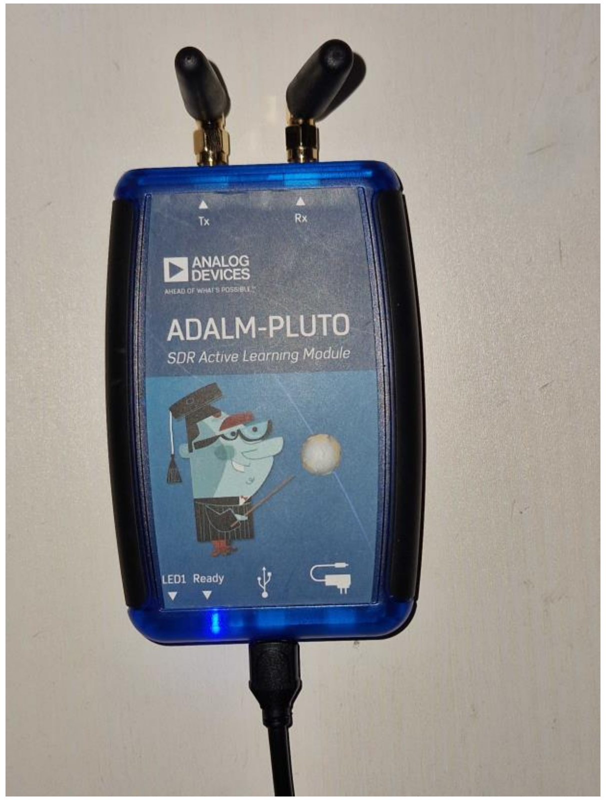
4.Doppler Radar with GNU Radio

▶ **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.

4.Doppler Radar with GNU Radio

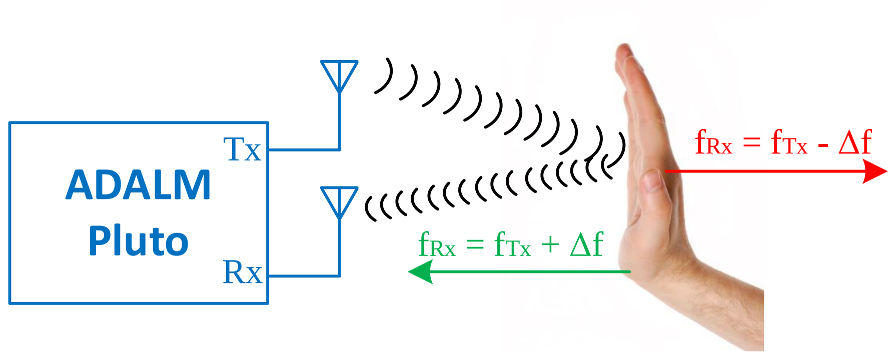
- **Connect the provided antennas to Rx and Tx of Pluto as depicted in the below picture and connect the Pluto to the PC using the USB Cable provided.**



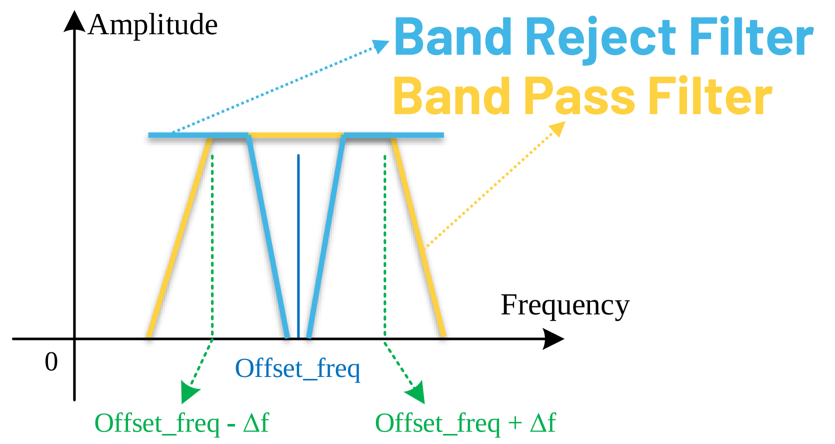
- **In the terminal, run the command "gnuradio-companion" to open GNU Radio Companion.**
- **After the application opens, go to File -> Open** ~/Desktop/ftc23_sdr/gnuradio/doppler_radar/doppler_radar.grc file

4.Doppler Radar with GNU Radio

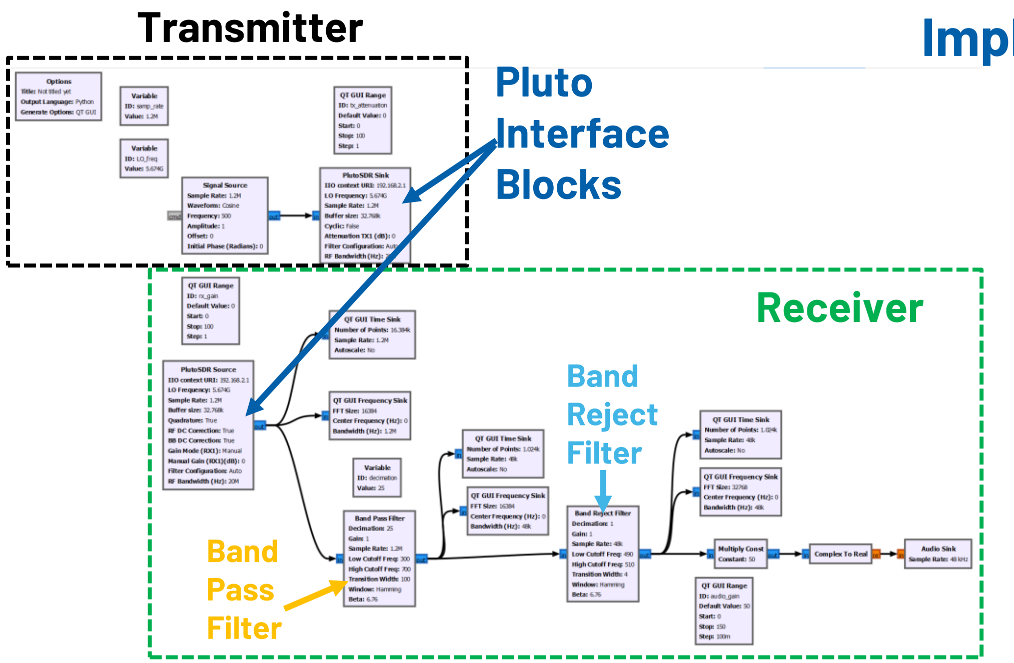
Theory

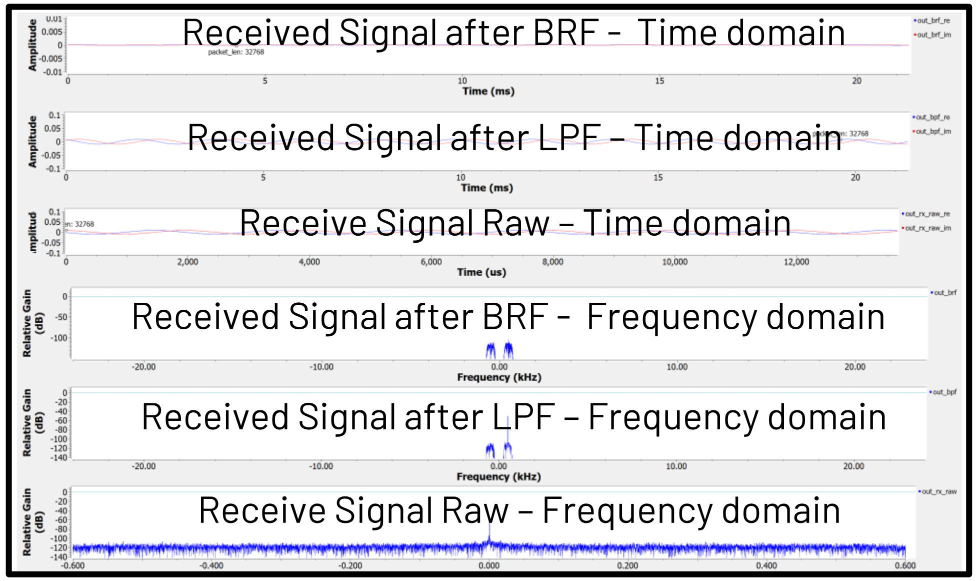


$$\Delta f = f\,\frac{2\cdot v}{c}$$



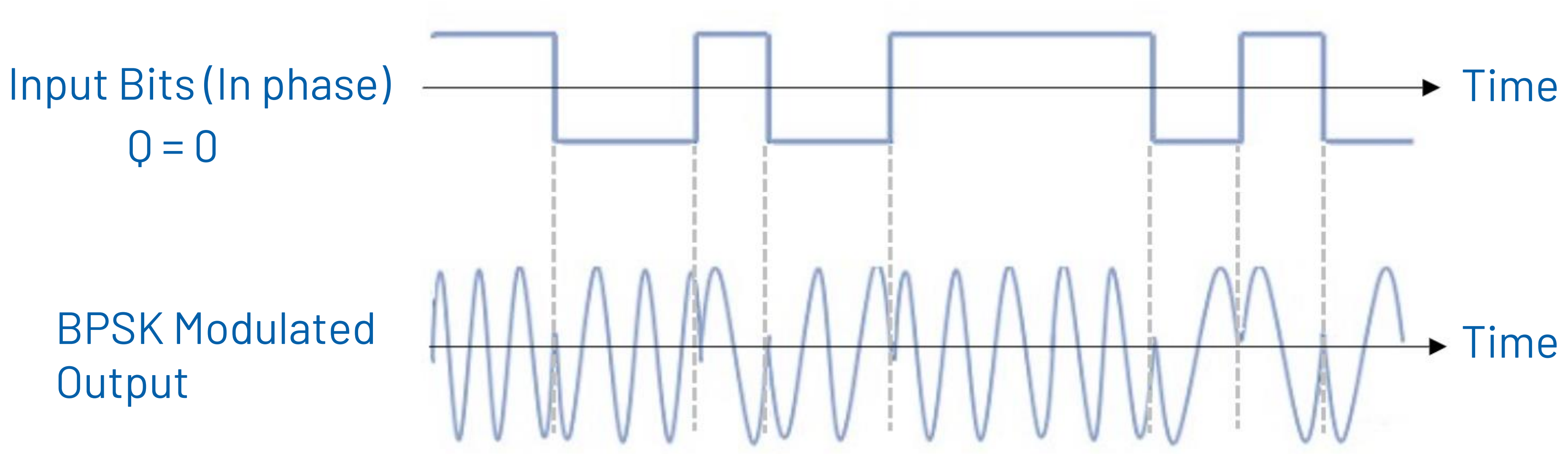
Implementation



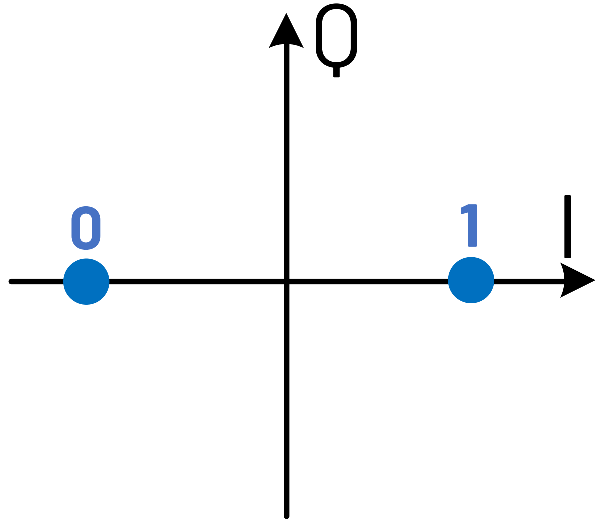


5.Binary Shift Keying in Python

- **Binary Shift Keying is a type of Phase Modulation where the symbols transmitted are 0 and 1**
- **What is Phase Modulation?**



BPSK Constellation:



Symbols: 1 -> 1 + 0j

0 -> -1 + 0j

5.Binary Shift Keying in Python

- **Where is Phase Modulation used?**
  - GSM
  - Satellite Television
  - Wi-Fi
  - Many Others

5.Binary Shift Keying in Python

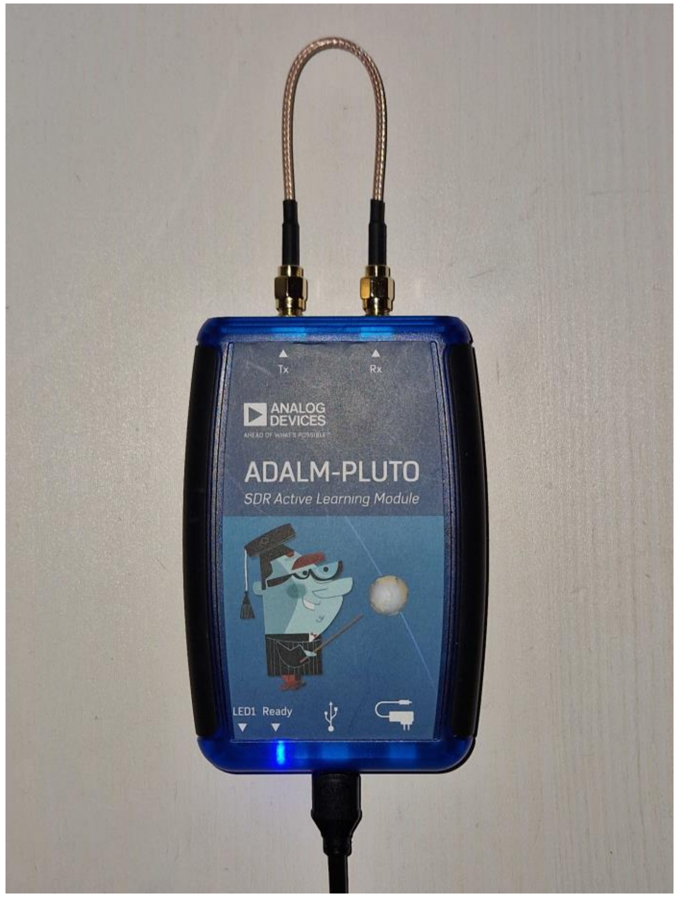
- **What are the problems when demodulating PSK signals?**
  - Phase Offset of LO at receiver
  - Frequency Offset of LO at receiver
  - Variation of these two with time, distance and temperature

(as we saw in the last application when frequency was varying with the change of distance)

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

5.Binary Shift Keying in Python

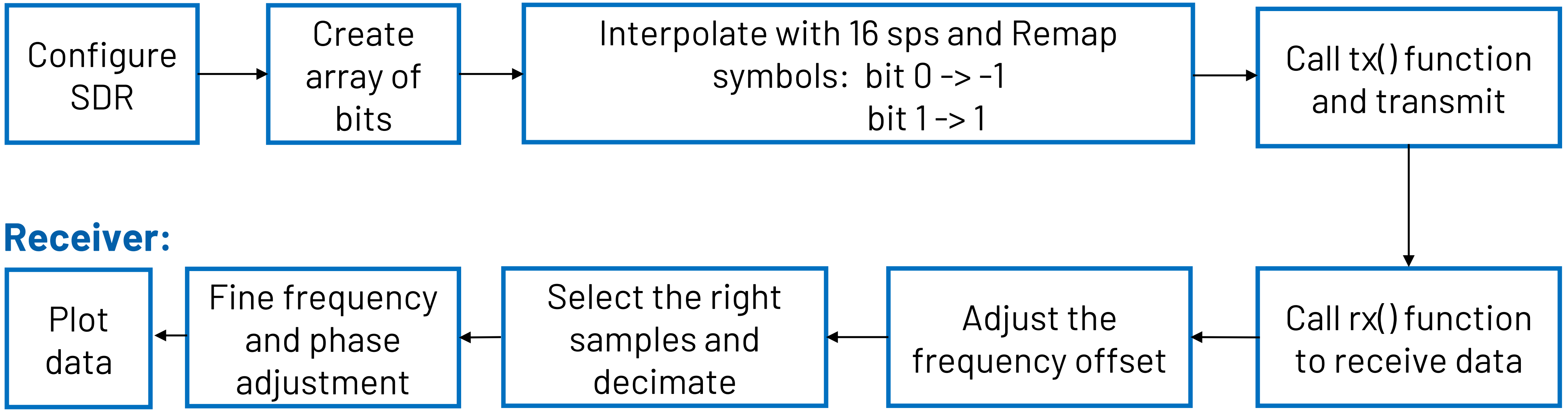
▶ **Connect Rx and Tx using the SMA cable from the kit (as in the picture from below) and connect the Pluto to the PC using the USB Cable provided.**



▶ **Go to** ~/Desktop/python/bpsk_loopback/**bpsk_pluto_loopback.py and open with Thony Python IDE.**

5.Binary Shift Keying in Python

- **Open** path/**bpsk_pluto_loopback.py with Thony IDE:**
- **What this code does?**
- **Transmitter:**



Configure SDR
Create array of bits
Interpolate with 16 sps and Remap symbols: bit 0 -> -1 bit 1 -> 1
Call tx() function and transmit
Call rx() function to receive data
Adjust the frequency offset
Select the right samples and decimate
Fine frequency and phase adjustment
Plot data

- **Receiver:**

5.Binary Shift Keying in Python

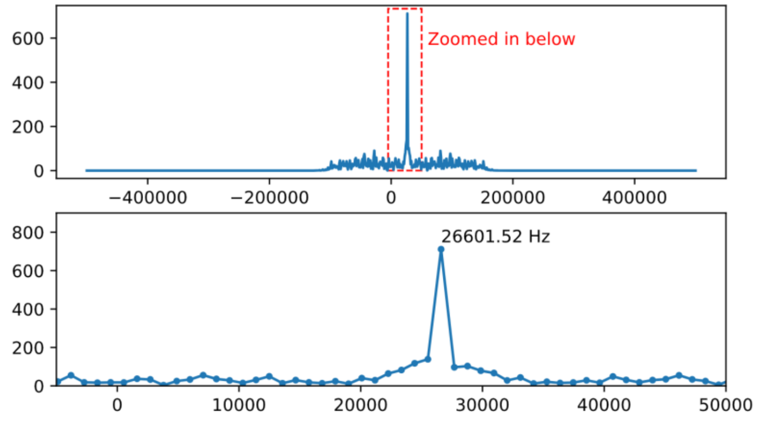
- **How we adjust the frequency offset?**
- **First square the received signal => all symbols (s(t))^2 will have a constant positive value**

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

```python
# Coarse frequancy adjustment
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**

```python
    # Coarse frequancy adjustment
    samples_adjust = samples**2 # square the received signal to obtain 2* offset frequency
    psd = np.fft.fftshift(np.abs(np.fft.fft(samples_adjust)))
    f = np.linspace(-fs/2.0, fs/2.0, len(psd))
    max_freq = f[np.argmax(psd)]
    Ts = 1/fs # calc sample period
    t = np.arange(0, Ts*len(samples), Ts) # create time vector
    samples = samples * np.exp(-1j*2*np.pi*max_freq*t/2.0)
```



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

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

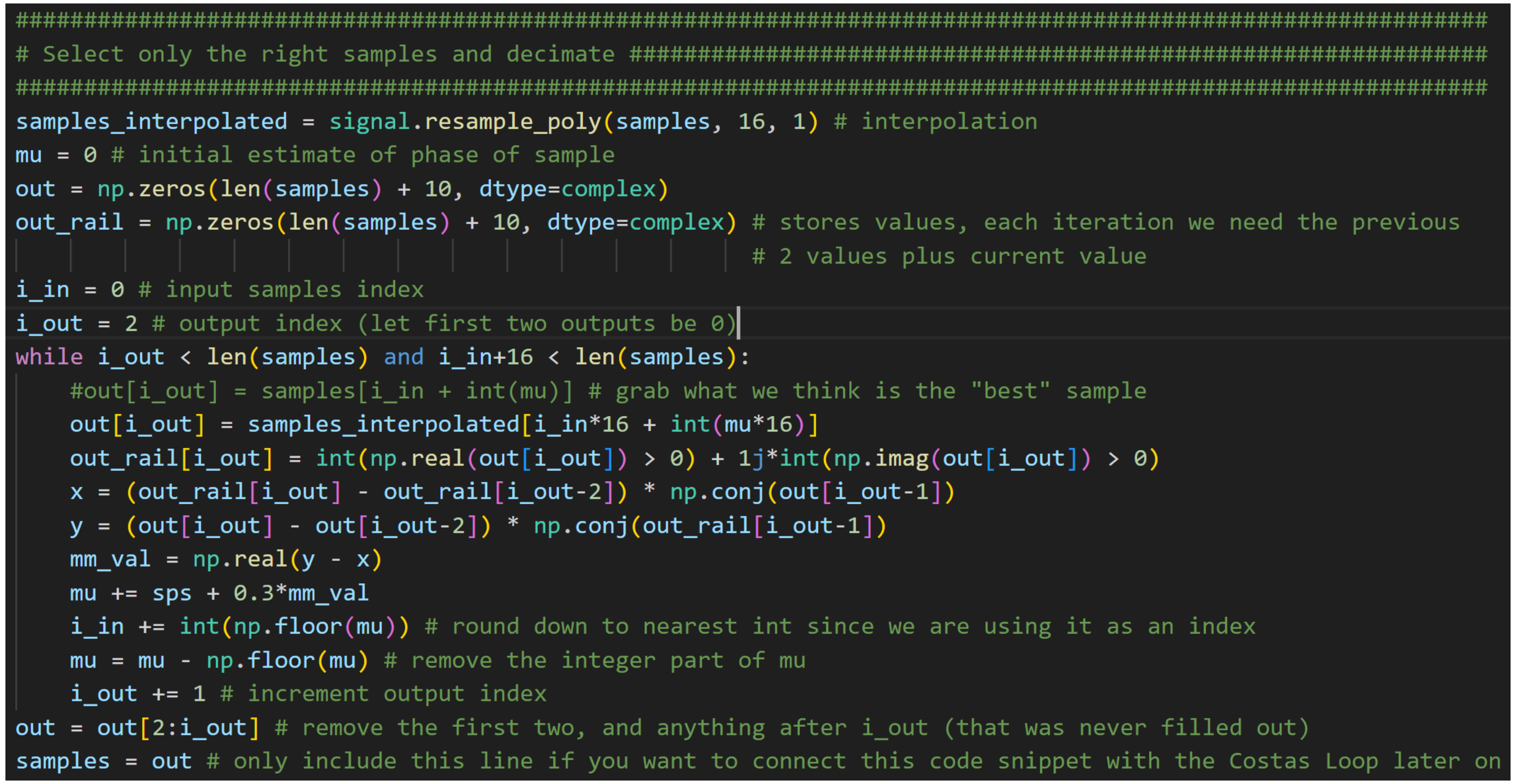
max_freq is the frequency where the peak power was measured

5.Binary Shift Keying in Python

- **How do we select the right samples (Mueller and Muller clock recovery technique [ ])?**

- **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 => we have to shift the input by 2.43 samples.**

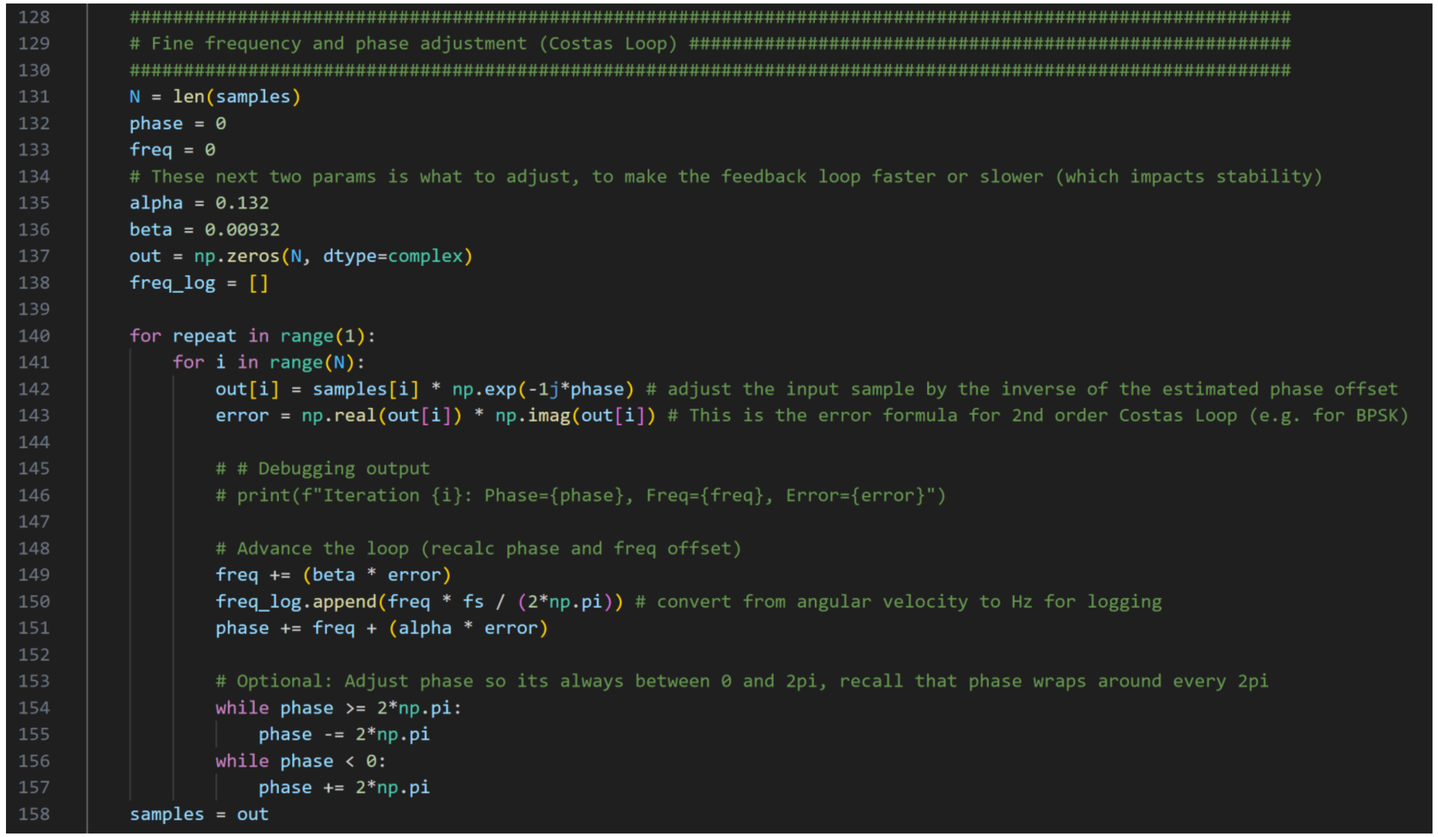
- **After a few iterations of the while loop, “mu” stabilizes and only the correct samples should be pulled.**



```
############################################################################################################
# Select only the right samples and decimate ###############################################################
############################################################################################################
samples_interpolated = signal.resample_poly(samples, 16, 1) # interpolation
mu = 0 # initial estimate of phase of sample
out = np.zeros(len(samples) + 10, dtype=complex)
out_rail = np.zeros(len(samples) + 10, dtype=complex) # stores values, each iteration we need the previous
                                                      # 2 values plus current value
i_in = 0 # input samples index
i_out = 2 # output index (let first two outputs be 0)
while i_out < len(samples) and i_in+16 < len(samples):
    #out[i_out] = samples[i_in + int(mu)] # grab what we think is the "best" sample
    out[i_out] = samples_interpolated[i_in*16 + int(mu*16)]
    out_rail[i_out] = int(np.real(out[i_out]) > 0) + 1j*int(np.imag(out[i_out]) > 0)
    x = (out_rail[i_out] - out_rail[i_out-2]) * np.conj(out[i_out-1])
    y = (out[i_out] - out[i_out-2]) * np.conj(out_rail[i_out-1])
    mm_val = np.real(y - x)
    mu += sps + 0.3*mm_val
    i_in += int(np.floor(mu)) # round down to nearest int since we are using it as an index
    mu = mu - np.floor(mu) # remove the integer part of mu
    i_out += 1 # increment output index
out = out[2:i_out] # remove the first two, and anything after i_out (that was never filled out)
samples = out # only include this line if you want to connect this code snippet with the Costas Loop later on
```

5.Binary Shift Keying in Python

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

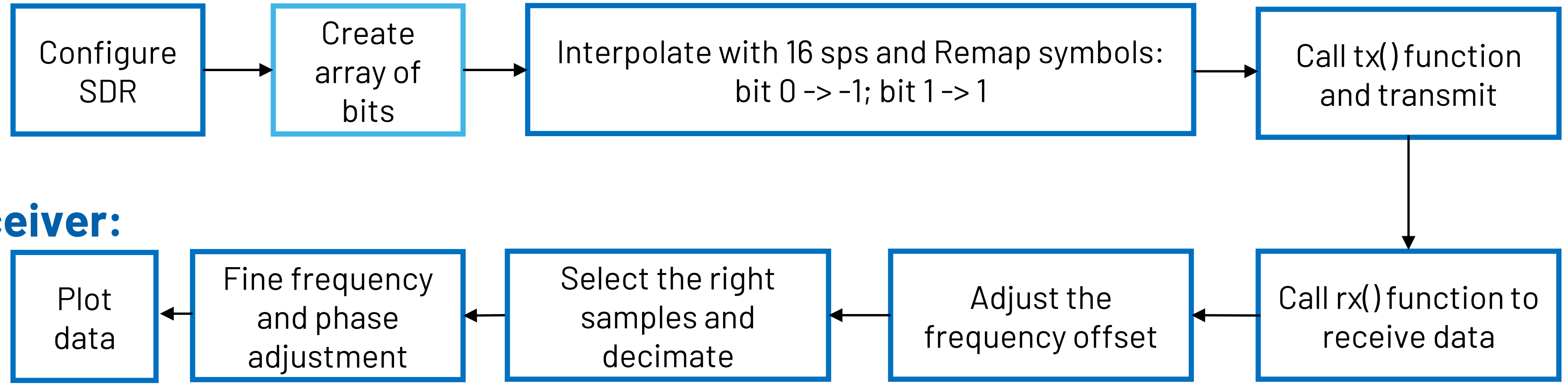


```
############################################################################################################
# Fine frequency and phase adjustment (Costas Loop) ########################################################
############################################################################################################
N = len(samples)
phase = 0
freq = 0
# These next two params is what to adjust, to make the feedback loop faster or slower (which impacts stability)
alpha = 0.132
beta = 0.00932
out = np.zeros(N, dtype=complex)
freq_log = []

for repeat in range(1):
    for i in range(N):
        out[i] = samples[i] * np.exp(-1j*phase) # adjust the input sample by the inverse of the estimated phase offset
        error = np.real(out[i]) * np.imag(out[i]) # This is the error formula for 2nd order Costas Loop (e.g. for BPSK)

        # # Debugging output
        # print(f"Iteration {i}: Phase={phase}, Freq={freq}, Error={error}")

        # Advance the loop (recalc phase and freq offset)
        freq += (beta * error)
        freq_log.append(freq * fs / (2*np.pi)) # convert from angular velocity to Hz for logging
        phase += freq + (alpha * error)

        # Optional: Adjust phase so its always between 0 and 2pi, recall that phase wraps around every 2pi
        while phase >= 2*np.pi:
            phase -= 2*np.pi
        while phase < 0:
            phase += 2*np.pi
samples = out
```

5.Binary Shift Keying in Python

▶ **Transmitter:**

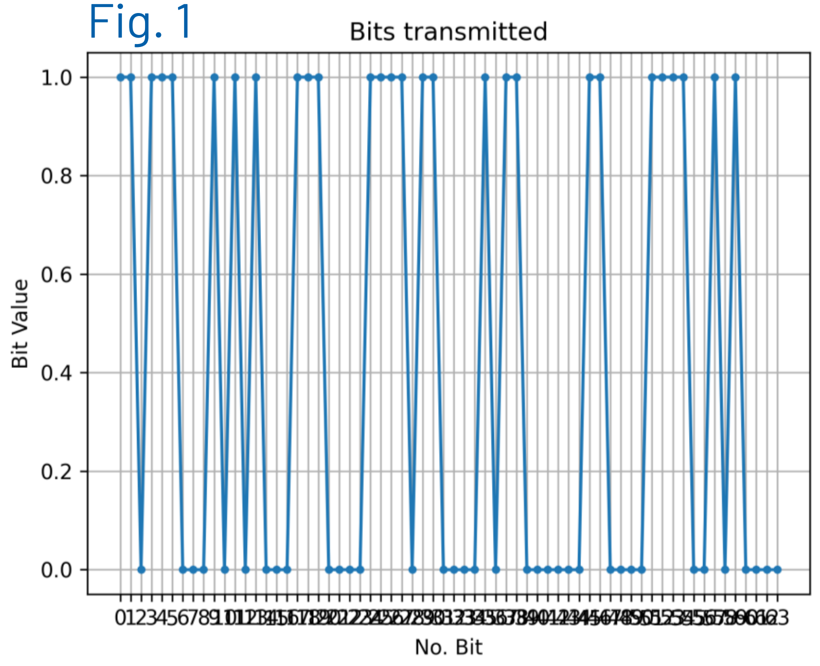
▶ **Receiver:**



Configure SDR
Create array of bits
Interpolate with 16 sps and Remap symbols: bit 0 -> -1; bit 1 -> 1
Call tx() function and transmit
Call rx() function to receive data
Adjust the frequency offset
Select the right samples and decimate
Fine frequency and phase adjustment
Plot data

▶ **Create the array of bits (this is the data you want to transmit):**

Fig. 1



5.Binary Shift Keying in Python

- **Transmitter:**
- **Receiver:**

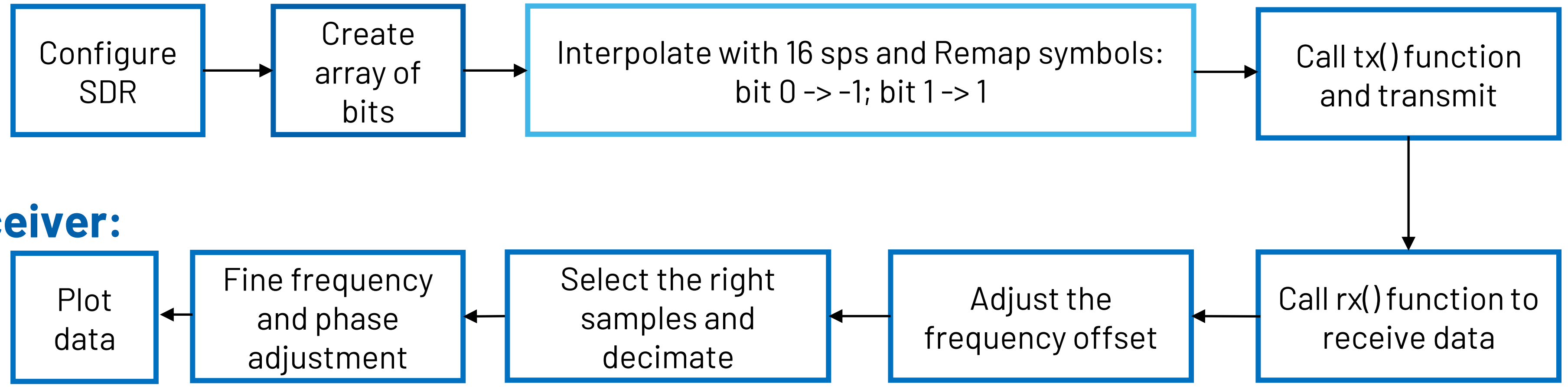

Configure SDR
Create array of bits
Interpolate with 16 sps and Remap symbols: bit 0 -> -1; bit 1 -> 1
Call tx() function and transmit
Call rx() function to receive data
Adjust the frequency offset
Select the right samples and decimate
Fine frequency and phase adjustment
Plot data

Fig. 2

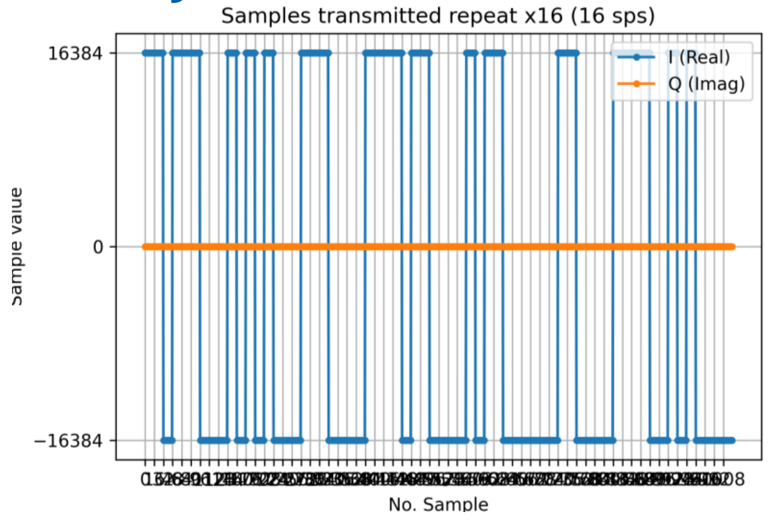
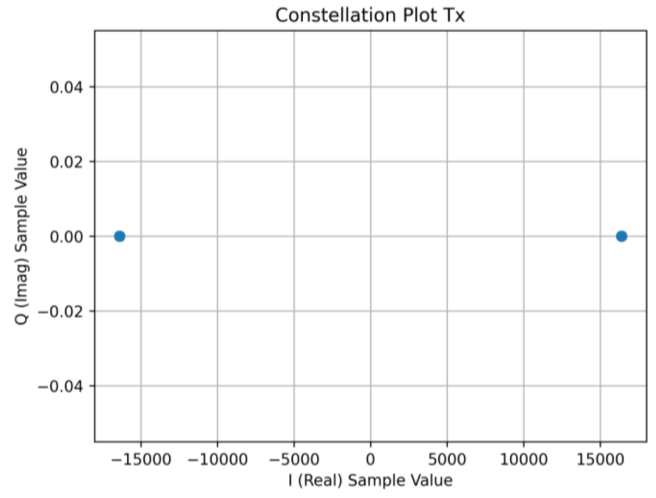


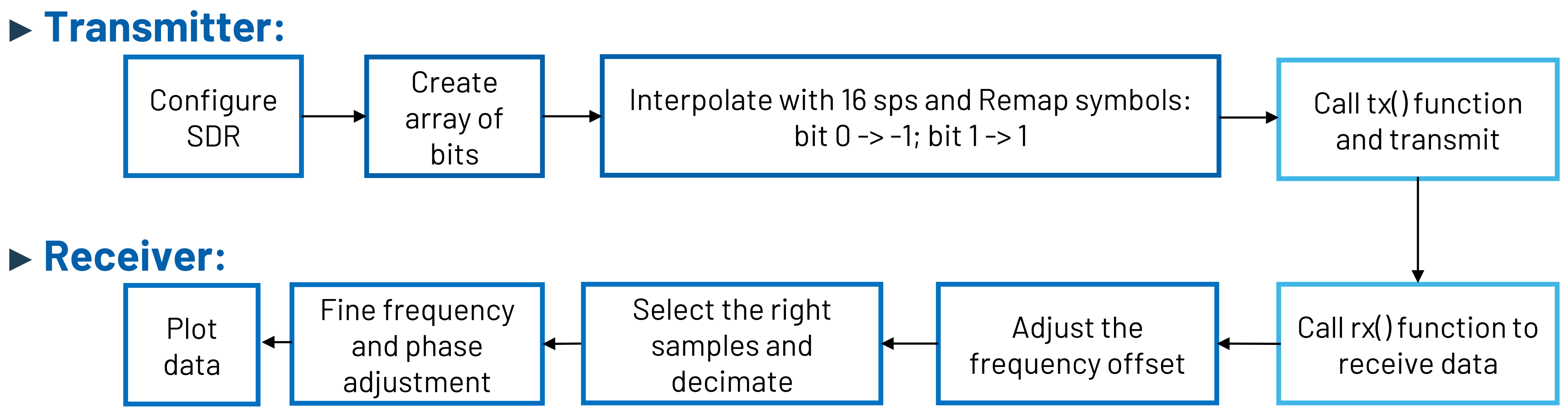
Fig. 3



- **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).

5. Binary Shift Keying in Python


▶ Transmitter:
Configure SDR
Create array of bits
Interpolate with 16 sps and Remap symbols: bit 0 -> -1; bit 1 -> 1
Call tx() function and transmit
▶ Receiver:
Call rx() function to receive data
Adjust the frequency offset
Select the right samples and decimate
Fine frequency and phase adjustment
Plot data

- **This is what we get after receive.**
- **Observe the phase offset and frequency offset that needs to be corrected, Q != 0.**

Fig. 4

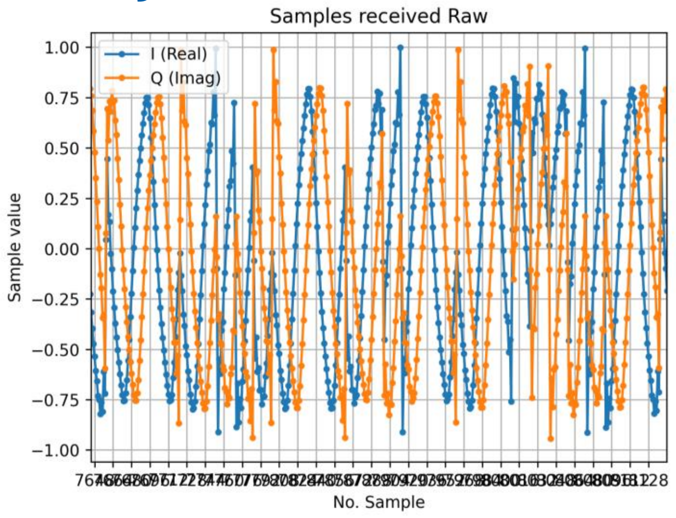
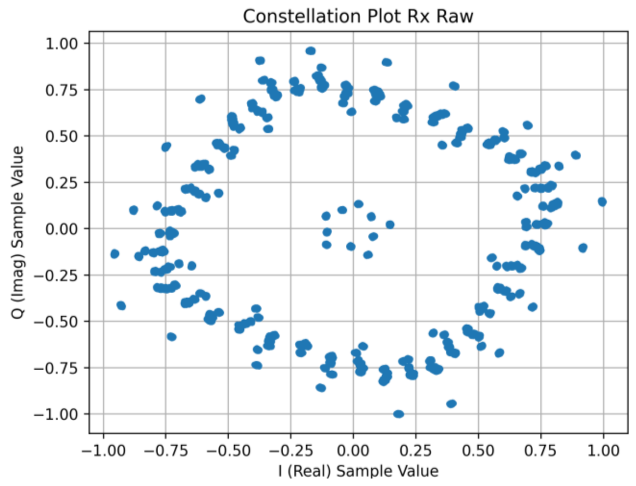
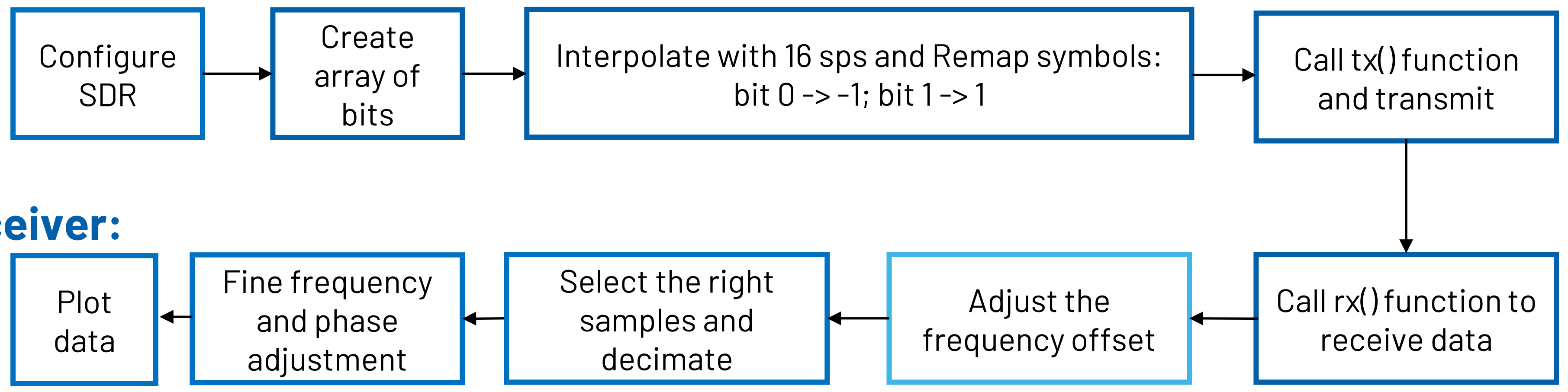


Fig. 5



5.Binary Shift Keying in Python

- **Transmitter:**


Configure SDR
Create array of bits
Interpolate with 16 sps and Remap symbols: bit 0 -> -1; bit 1 -> 1
Call tx() function and transmit
Call rx() function to receive data
Adjust the frequency offset
Select the right samples and decimate
Fine frequency and phase adjustment
Plot data

- **Receiver:**

Fig. 6

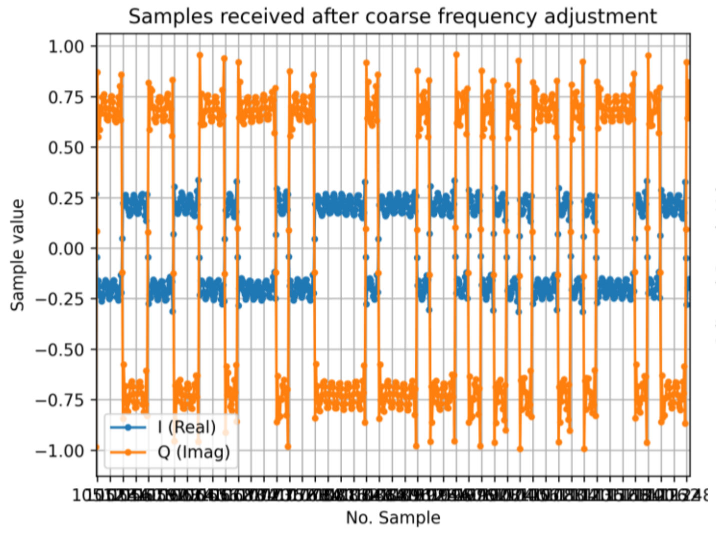
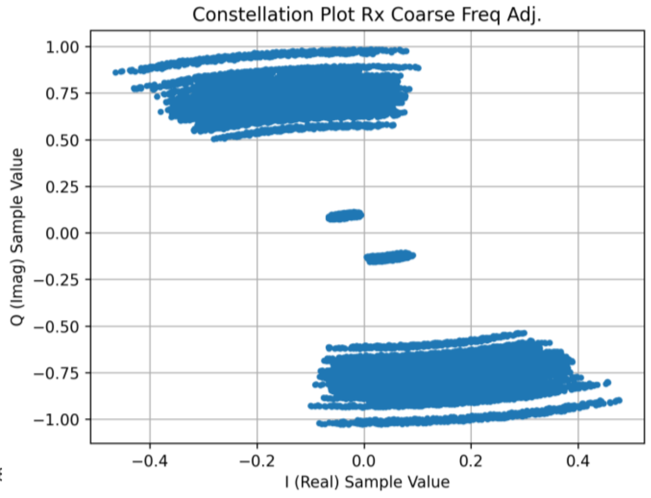


Fig. 7



- **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 != 0**

5.Binary Shift Keying in Python

- **Transmitter:**
- **Receiver:**

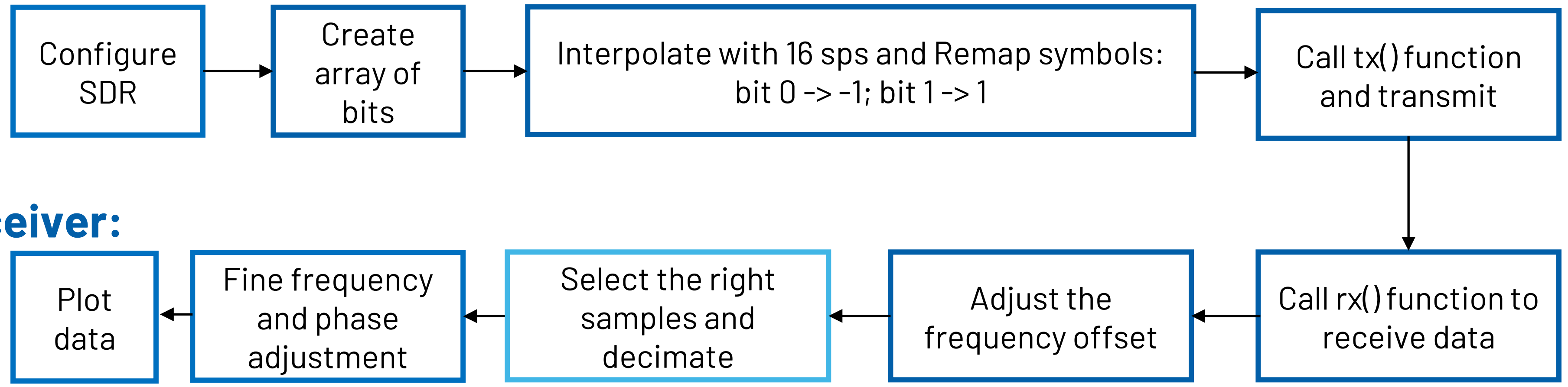

Configure SDR
Create array of bits
Interpolate with 16 sps and Remap symbols: bit 0 -> -1; bit 1 -> 1
Call tx() function and transmit
Call rx() function to receive data
Adjust the frequency offset
Select the right samples and decimate
Fine frequency and phase adjustment
Plot data

Fig. 8

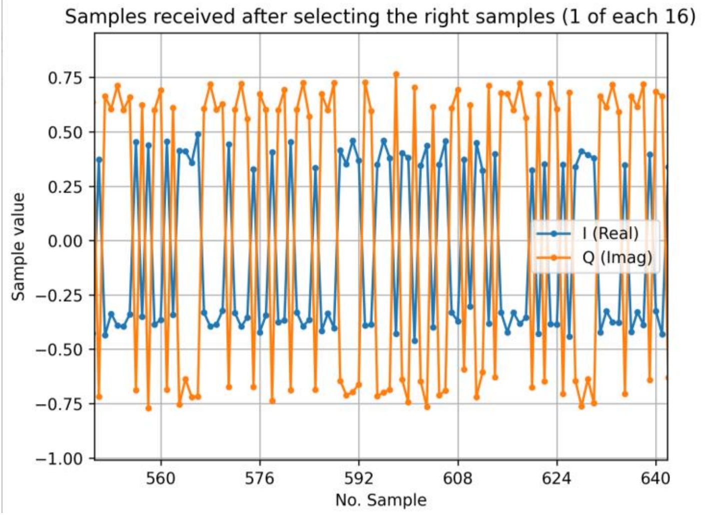
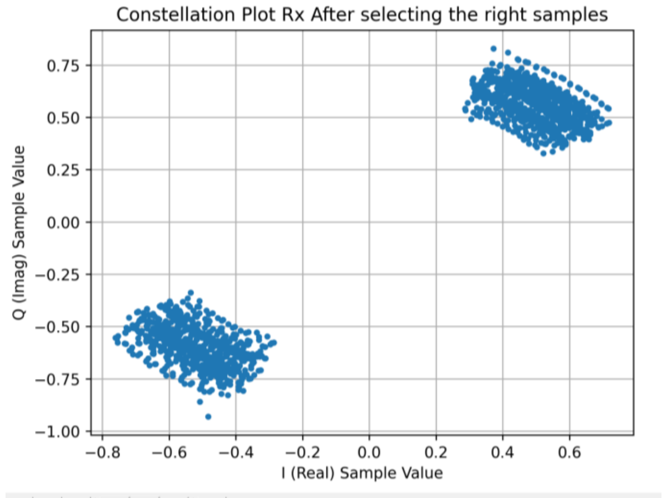


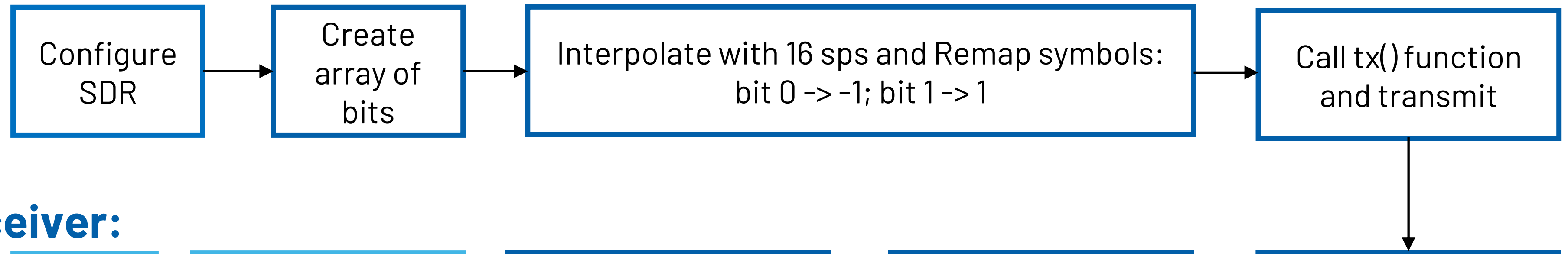
Fig. 9



- **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 != 0**

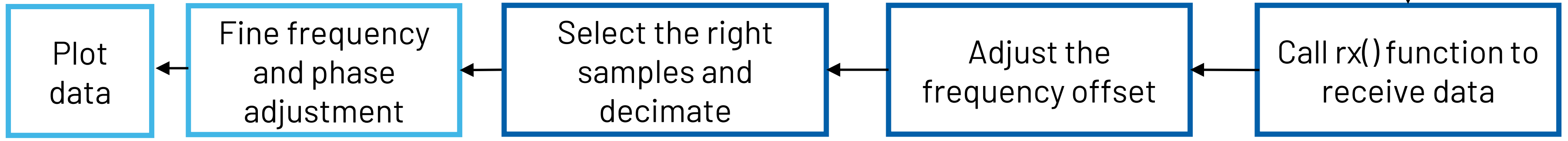
5.Binary Shift Keying in Python

- **Transmitter:**



Configure SDR
Create array of bits
Interpolate with 16 sps and Remap symbols: bit 0 -> -1; bit 1 -> 1
Call tx() function and transmit

- **Receiver:**



Call rx() function to receive data
Adjust the frequency offset
Select the right samples and decimate
Fine frequency and phase adjustment
Plot data

- **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.**

Fig. 11

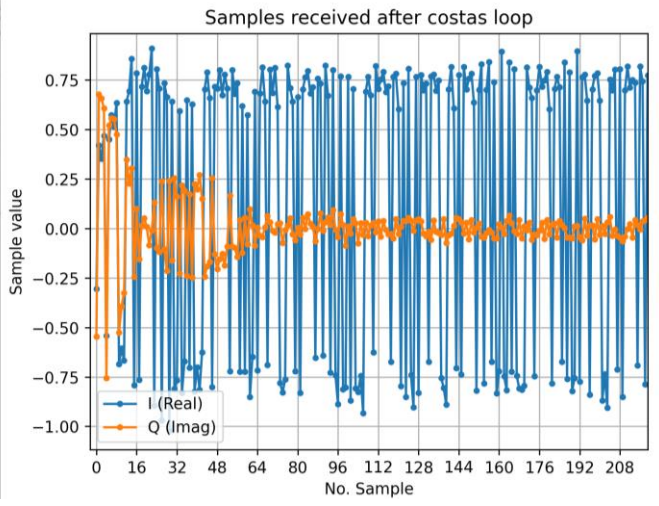
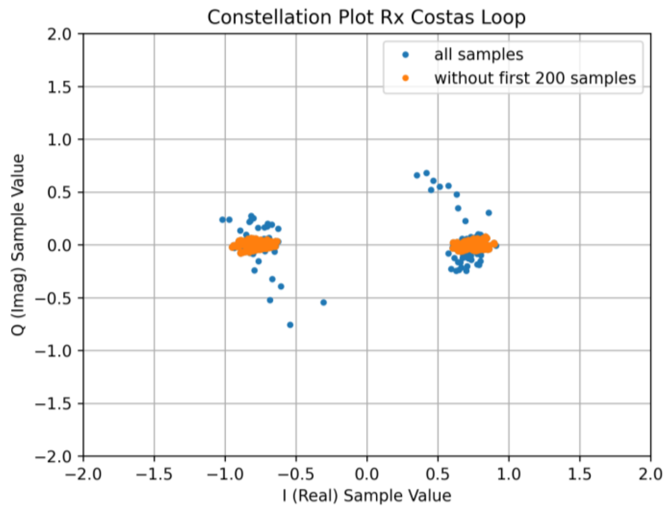


Fig. 12



5. Binary Shift Keying in Python

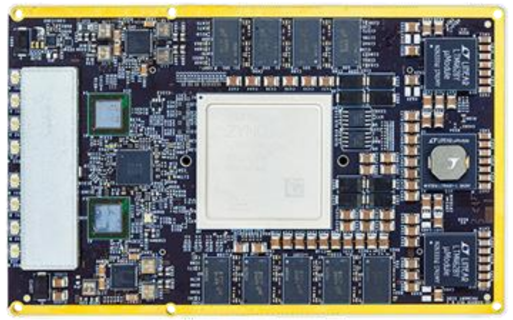
▶ **Using PyADI-IIO you can reuse the code and run it on more expensive SDRs like Jupyter and Talise.**

PyADI-IIO

PYTHON FOR ADI INDUSTRIAL I/O DEVICES

ADALM-PLUTO
- AD9363
- Zynq®-7010

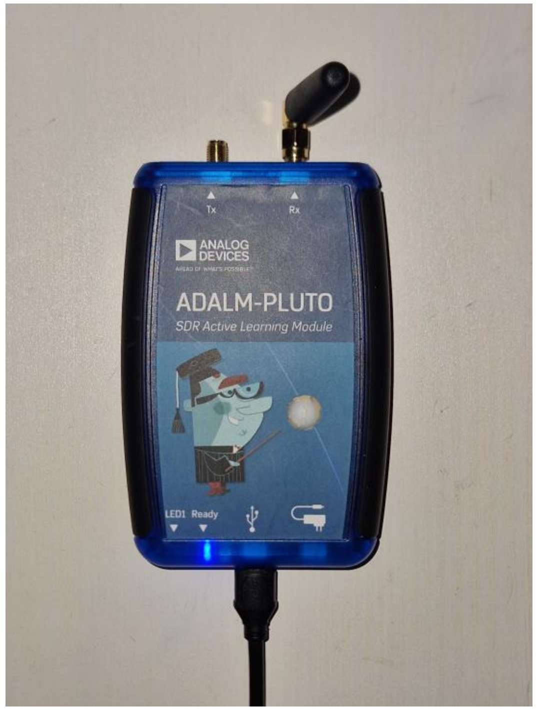
Jupiter
- ADRV9002
- ZynqMP-ZU3EG



ADRV9009-ZU11EG
- 2× ADRV9009
- ZynqMP-ZU11EG

6.Quadrature Shift Keying in GNU Radio

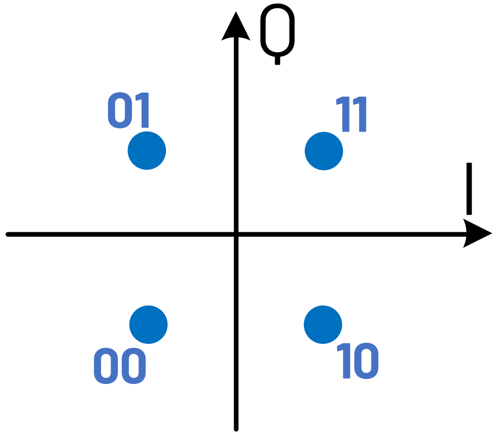
- **Connect the provided antennas to Rx of Pluto as depicted in the below picture and connect the Pluto to the PC using the USB Cable provided.**



- **In the terminal, run the command “gnuradio-companion” to open GNU Radio Companion.**
- **After the application opens, go to File -> Open** add path to qpsk_point_to_point_rx_console.grc file

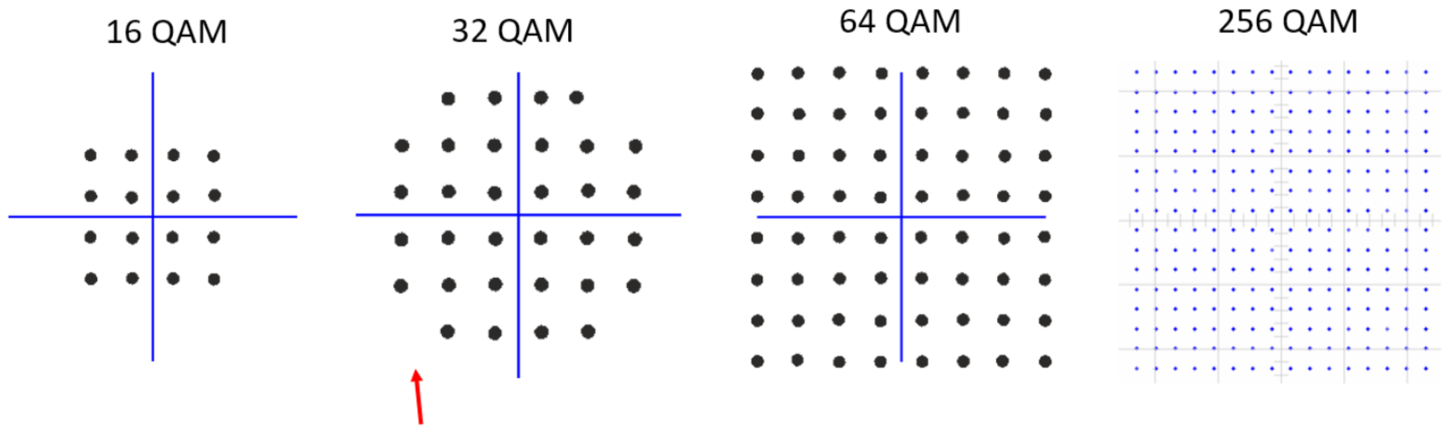
6.Quadrature Shift Keying in GNU Radio

- **QPSK is similar with BPSK but has more symbols.**



Symbols:	00 -> 0 + 0j
	01 -> -1 + 1j
	10 -> 1 – 1j
	11 -> 1 + 1j

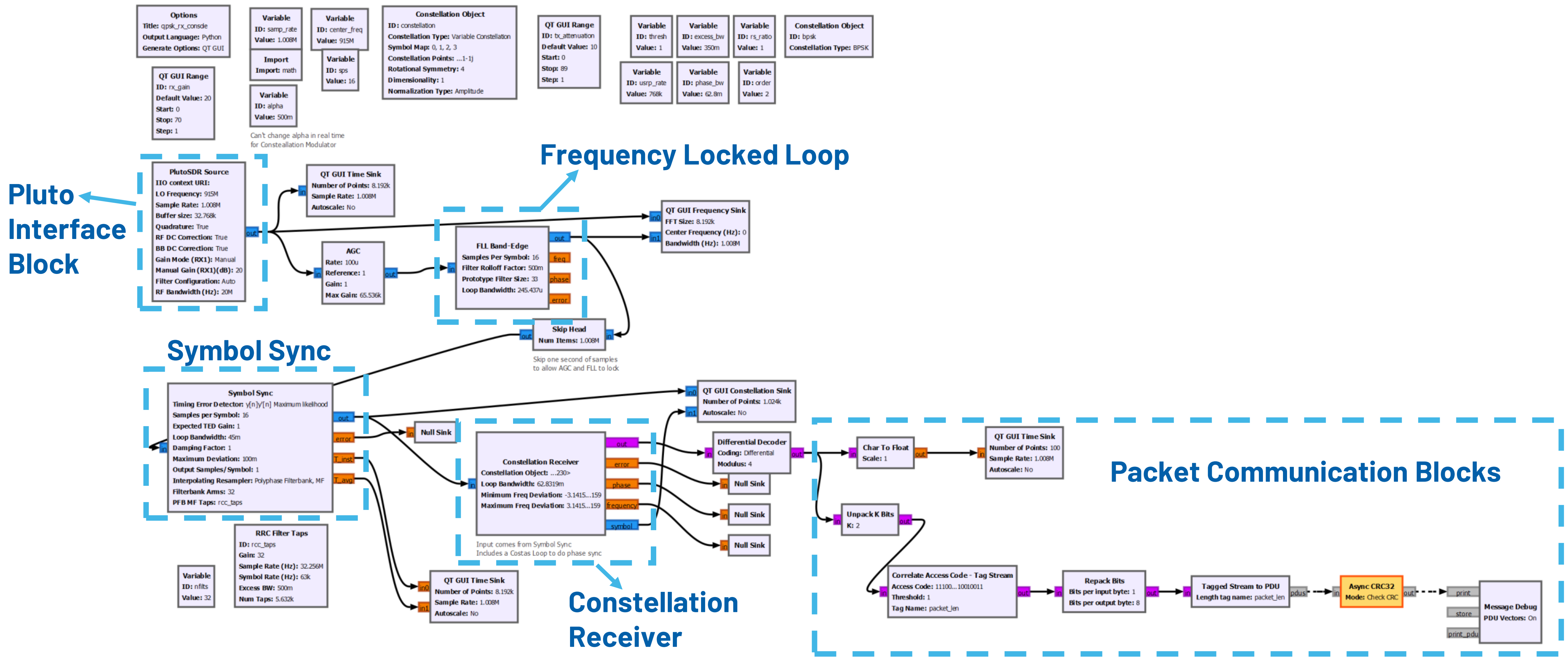
- **PSK with more than 4 symbols (here the amplitude is modulated too):**



QAM doesn't have to be a square

6.Quadrature Shift Keying in GNU Radio

- After opening qpsk_point_to_point_rx_console.grc file in GNU Radio Companion you should see this flowgraph:



Options
Title: qpsk_rx_consde
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 1.008M

Variable
ID: center_freq
Value: 915M

Import
Import: math

Variable
ID: sps
Value: 16

Variable
ID: alpha
Value: 500m

Can't change alpha in real time
for Consteallation Modulator

QT GUI Range
ID: rx_gain
Default Value: 20
Start: 0
Stop: 70
Step: 1

Constellation Object
ID: constellation
Constellation Type: Variable Constellation
Symbol Map: 0, 1, 2, 3
Constellation Points: ...1-1j
Rotational Symmetry: 4
Dimensionality: 1
Normalization Type: Amplitude

QT GUI Range
ID: tx_attenuation
Default Value: 10
Start: 0
Stop: 89
Step: 1

Variable
ID: thresh
Value: 1

Variable
ID: excess_bw
Value: 350m

Variable
ID: rs_ratio
Value: 1

Constellation Object
ID: bpsk
Constellation Type: BPSK

Variable
ID: usrp_rate
Value: 768k

Variable
ID: phase_bw
Value: 62.8m

Variable
ID: order
Value: 2

PlutoSDR Source
IIO context URI:
LO Frequency: 915M
Sample Rate: 1.008M
Buffer size: 32.768k
Quadrature: True
RF DC Correction: True
BB DC Correction: True
Gain Mode (RX1): Manual
Manual Gain (RX1)(dB): 20
Filter Configuration: Auto
RF Bandwidth (Hz): 20M

QT GUI Time Sink
Number of Points: 8.192k
Sample Rate: 1.008M
Autoscale: No

AGC
Rate: 100u
Reference: 1
Gain: 1
Max Gain: 65.536k

FLL Band-Edge
Samples Per Symbol: 16
Filter Rolloff Factor: 500m
Prototype Filter Size: 33
Loop Bandwidth: 245.437u

QT GUI Frequency Sink
FFT Size: 8.192k
Center Frequency (Hz): 0
Bandwidth (Hz): 1.008M

Skip Head
Num Items: 1.008M

Skip one second of samples
to allow AGC and FLL to lock

Symbol Sync
Timing Error Detector: y[n]y'[n] Maximum likelihood
Samples per Symbol: 16
Expected TED Gain: 1
Loop Bandwidth: 45m
Damping Factor: 1
Maximum Deviation: 100m
Output Samples/Symbol: 1
Interpolating Resampler: Polyphase Filterbank, MF
Filterbank Arms: 32
PFB MF Taps: rcc_taps

Null Sink

QT GUI Constellation Sink
Number of Points: 1.024k
Autoscale: No

Constellation Receiver
Constellation Object: ...230>
Loop Bandwidth: 62.8319m
Minimum Freq Deviation: -3.1415...159
Maximum Freq Deviation: 3.1415...159

Input comes from Symbol Sync
Includes a Costas Loop to do phase sync

Differential Decoder
Coding: Differential
Modulus: 4

Null Sink

Null Sink

Null Sink

RRC Filter Taps
ID: rcc_taps
Gain: 32
Sample Rate (Hz): 32.256M
Symbol Rate (Hz): 63k
Excess BW: 500m
Num Taps: 5.632k

Variable
ID: nfilts
Value: 32

QT GUI Time Sink
Number of Points: 8.192k
Sample Rate: 1.008M
Autoscale: No

Char To Float
Scale: 1

QT GUI Time Sink
Number of Points: 100
Sample Rate: 1.008M
Autoscale: No

Unpack K Bits
K: 2

Correlate Access Code - Tag Stream
Access Code: 11100...10010011
Threshold: 1
Tag Name: packet_len

Repack Bits
Bits per input byte: 1
Bits per output byte: 8

Tagged Stream to PDU
Length tag name: packet_len

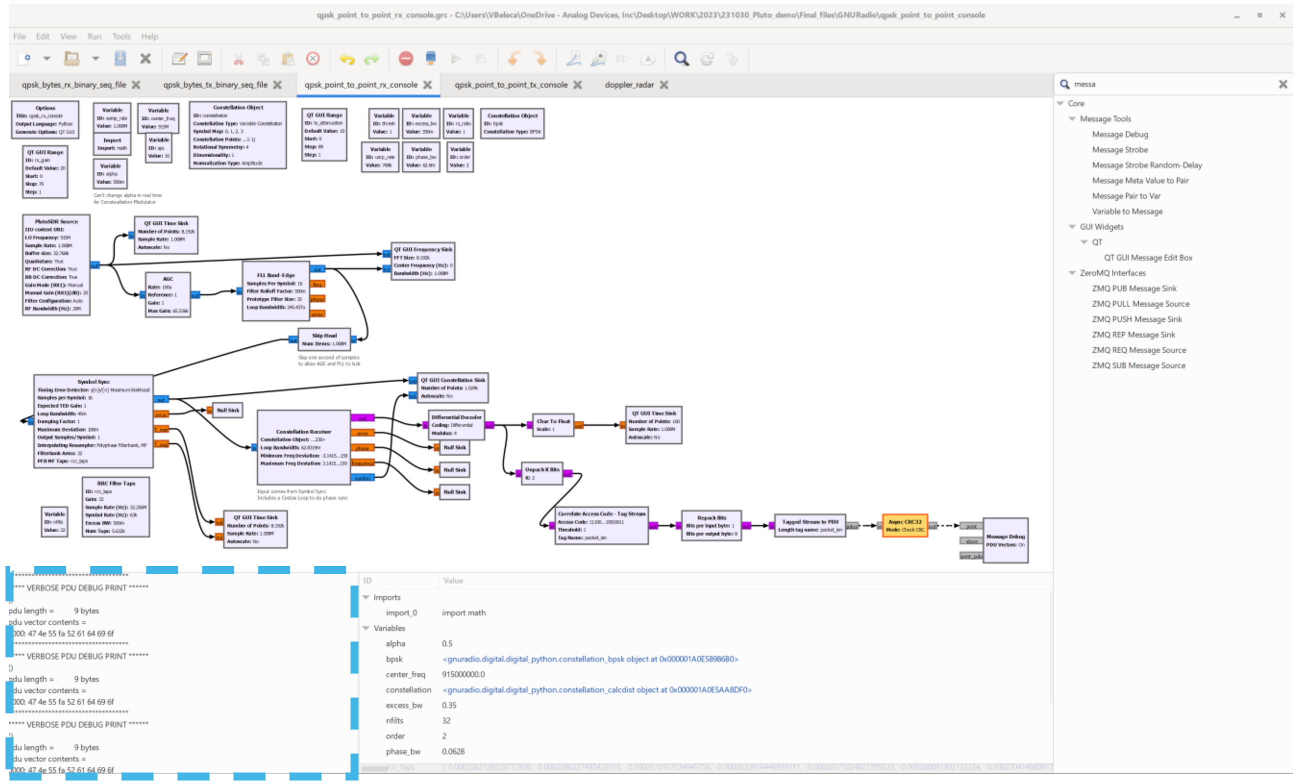
Async CRC32
Mode: Check CRC

Message Debug
PDU Vectors: On

- We will send you a set of hex values and you can receive them using your Pluto.

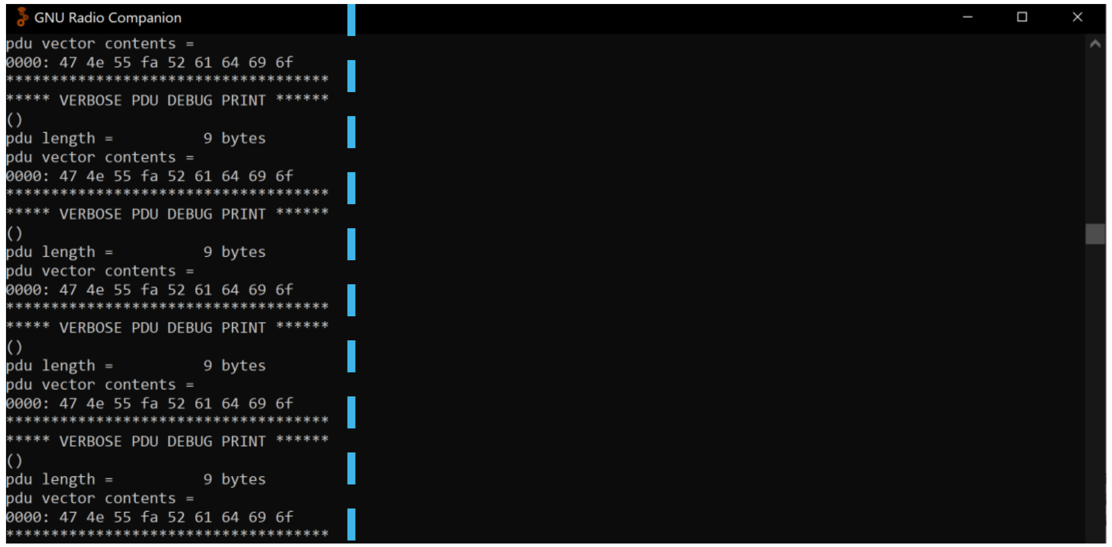
6.Quadrature Shift Keying in GNU Radio

- **After running the qpsk_point_to_point_rx_console.grc flowgraph, you should see the set of values we transmitted on the console in GNU Radio Companion and on the terminal, as shown below:**



```
*********************************
**** VERBOSE PDU DEBUG PRINT ******
()
pdu length =         9 bytes
pdu vector contents =
000: 47 4e 55 fa 52 61 64 69 6f
*********************************
**** VERBOSE PDU DEBUG PRINT ******
()
pdu length =         9 bytes
du vector contents =
000: 47 4e 55 fa 52 61 64 69 6f
*********************************
***** VERBOSE PDU DEBUG PRINT ******
()
du length =         9 bytes
du vector contents =
000: 47 4e 55 fa 52 61 64 69 6f
```

ID	Value
Imports	
import_0	import math
Variables	
alpha	0.5
bpsk	<gnuradio.digital.digital_python.constellation_bpsk object at 0x000001A0E58986B0>
center_freq	915000000.0
constellation	<gnuradio.digital.digital_python.constellation_calcdist object at 0x000001A0E5AA8DF0>
excess_bw	0.35
nfilts	32
order	2
phase_bw	0.0628

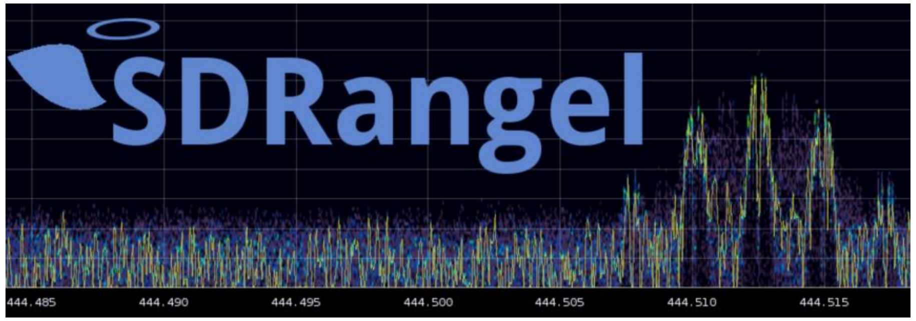


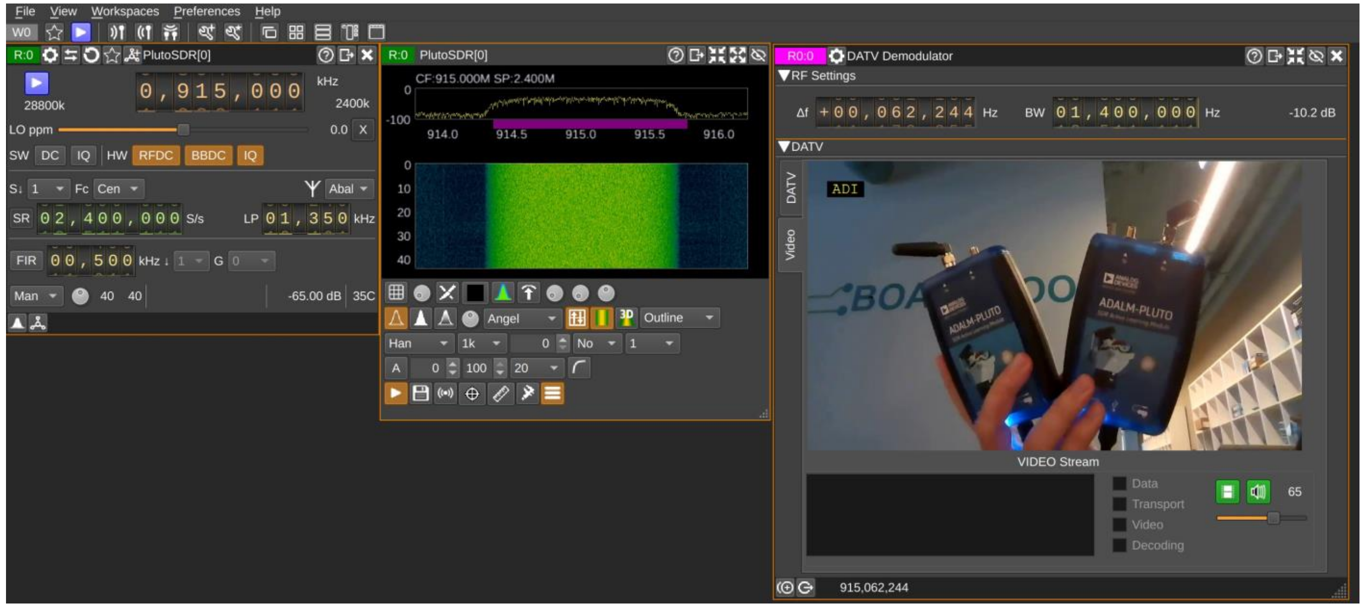
GNU Radio Companion

```
pdu vector contents =
0000: 47 4e 55 fa 52 61 64 69 6f
***********************************
***** VERBOSE PDU DEBUG PRINT ******
()
pdu length =         9 bytes
pdu vector contents =
0000: 47 4e 55 fa 52 61 64 69 6f
***********************************
***** VERBOSE PDU DEBUG PRINT ******
()
pdu length =         9 bytes
pdu vector contents =
0000: 47 4e 55 fa 52 61 64 69 6f
***********************************
***** VERBOSE PDU DEBUG PRINT ******
()
pdu length =         9 bytes
pdu vector contents =
0000: 47 4e 55 fa 52 61 64 69 6f
***********************************
***** VERBOSE PDU DEBUG PRINT ******
()
pdu length =         9 bytes
pdu vector contents =
0000: 47 4e 55 fa 52 61 64 69 6f
***********************************
```

7.Receiving QPSK modulated Video with SDRs

- **Using Open Source software such as SDRangel, more complex tasks can be achieved such as receiving and displaying DATV (Digital Amateur Television).**





7.Spectrum Paint

- **Gnu radio – fosphor**

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 (Pluto, Jupiter, Talise) 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/

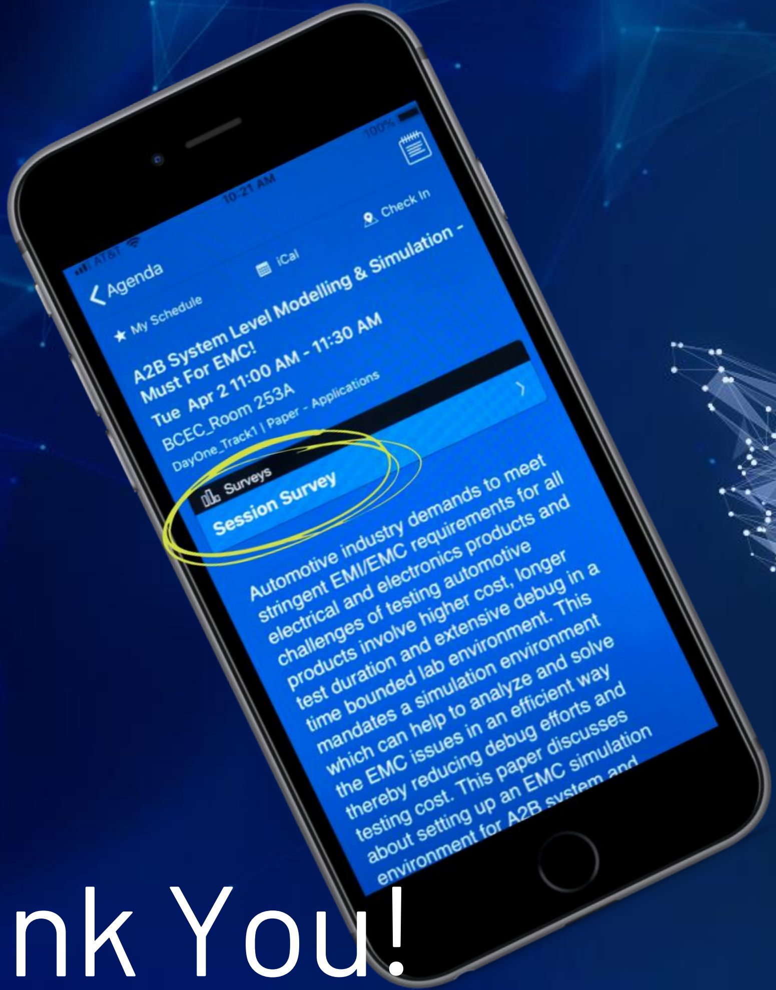
[8] J. Gallachio - https://github.com/gallicchio/learnSDR

[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



< Agenda

★ My Schedule

iCal

Check In

A2B System Level Modelling & Simulation - Must For EMC!

Tue Apr 2 11:00 AM - 11:30 AM

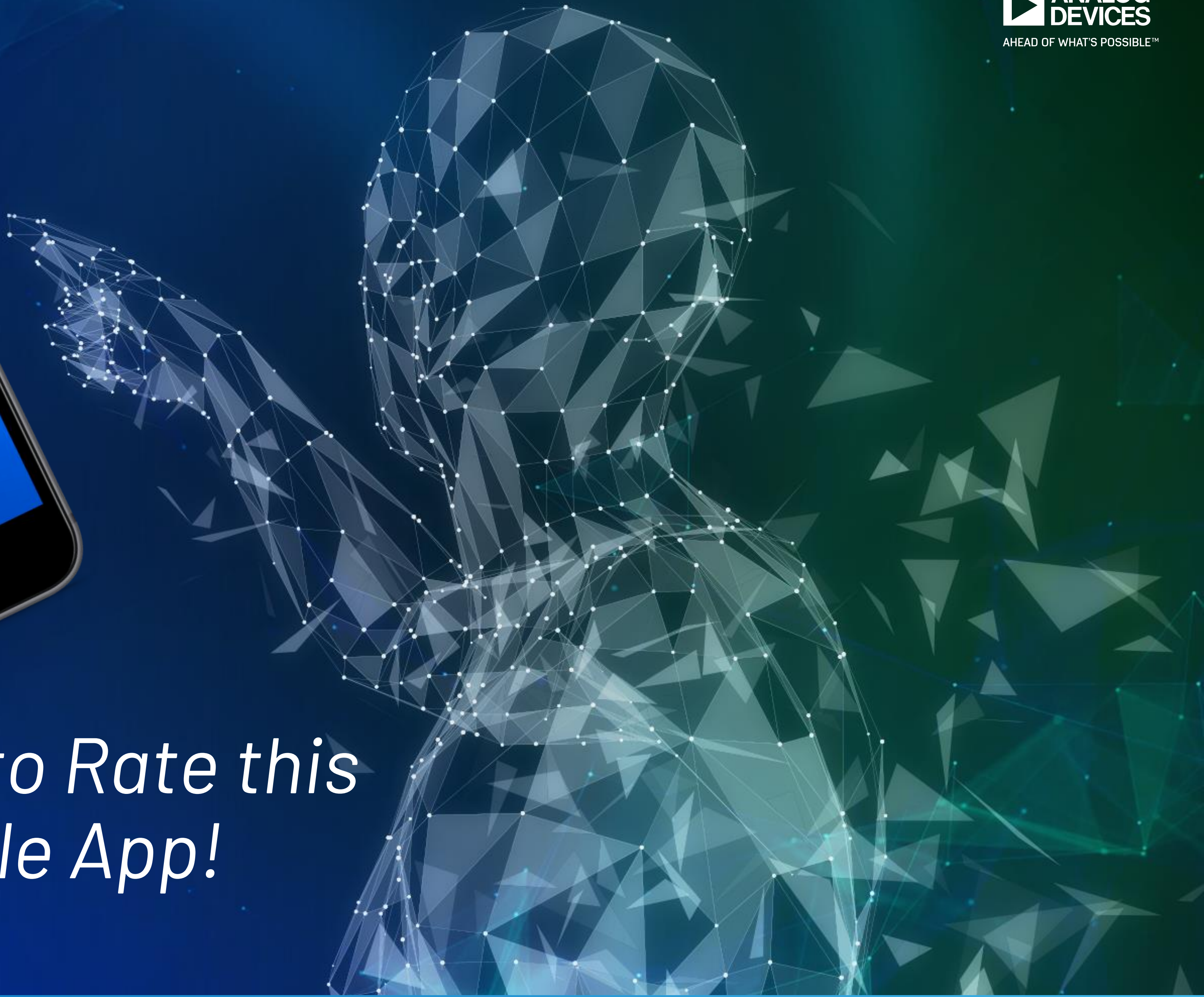
BCEC_Room 253A

DayOne_Track1 | Paper - Applications

Surveys

Session Survey

Automotive industry demands to meet stringent EMI/EMC requirements for all electrical and electronics products and challenges of testing automotive products involve higher cost, longer test duration and extensive debug in a time bounded lab environment. This mandates a simulation environment which can help to analyze and solve the EMC issues in an efficient way thereby reducing debug efforts and testing cost. This paper discusses about setting up an EMC simulation environment for A2B system and



Thank You!

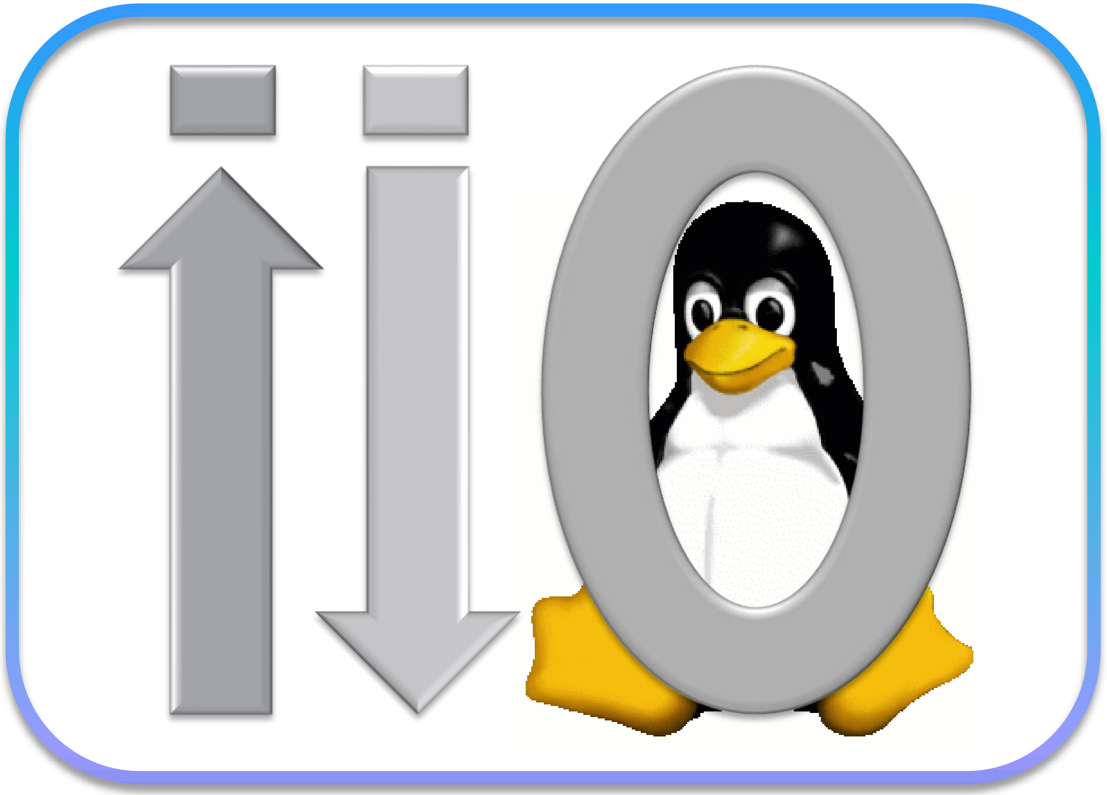
Please Remember to Rate this Session in the Mobile App!

Backup Slides

IIO Overview

What is IIO?

- Linux kernel **I**ndustrial **I**nput / **O**utput framework
  - Not really just for Industrial IO
  - All non-HID IO
  - ADC, DAC, light, accelerometer, gyro, magnetometer, humidity, temperature, pressure, rotation, angular momentum, chemical, health, proximity, counters, etc.
- In the upstream Linux kernel for more than 10 years.
- Mailing list:
  - linux-iio@vger.kernel.org



https://www.kernel.org/doc/html/latest/driver-api/iio/index.html

Why use IIO for high-speed converter systems & SDR?

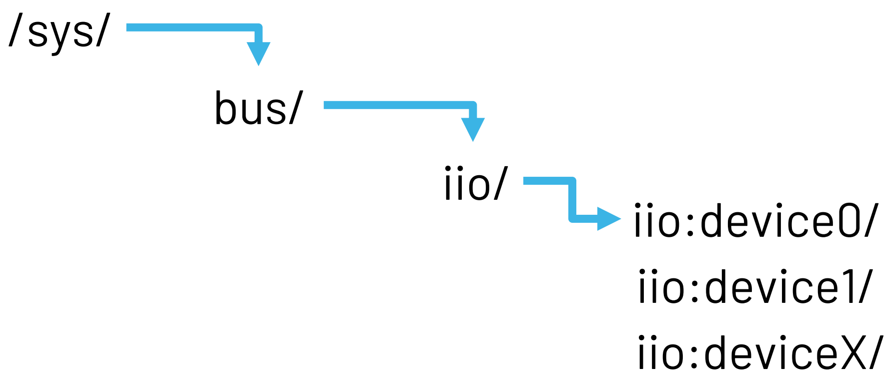
- Provides hardware abstraction layer
  - Allows sharing of infrastructure
  - Allows developers to focus on the solution
  - Allows application re-use

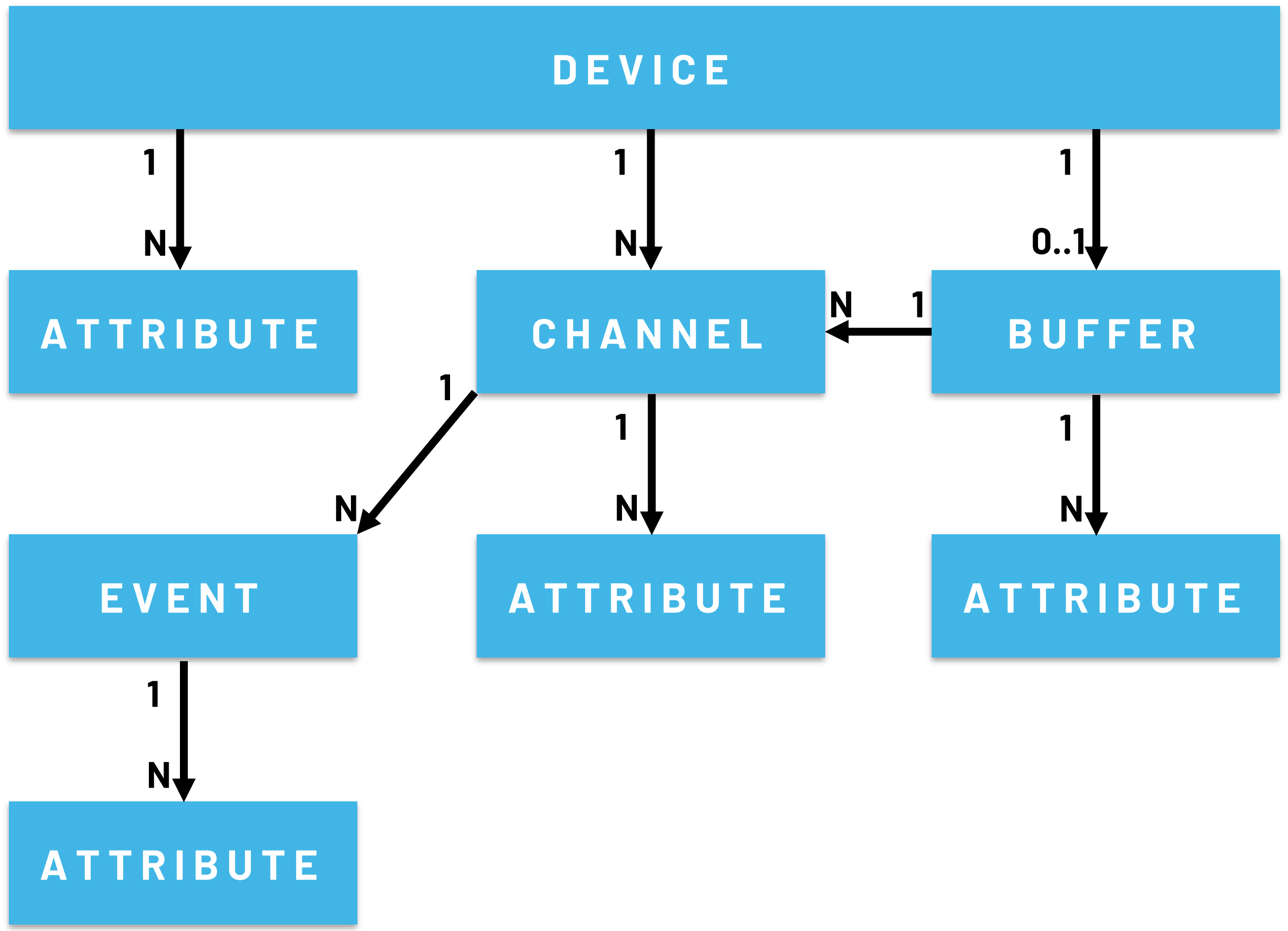
- Kernel drivers have low-level & low-latency access to hardware
  - MMIO
  - Interrupts
  - DMA
  - Memory

- IIO provides fast and efficient data transport
  - From device to application
  - From application to device
  - From device to network/storage

IIO – Devices

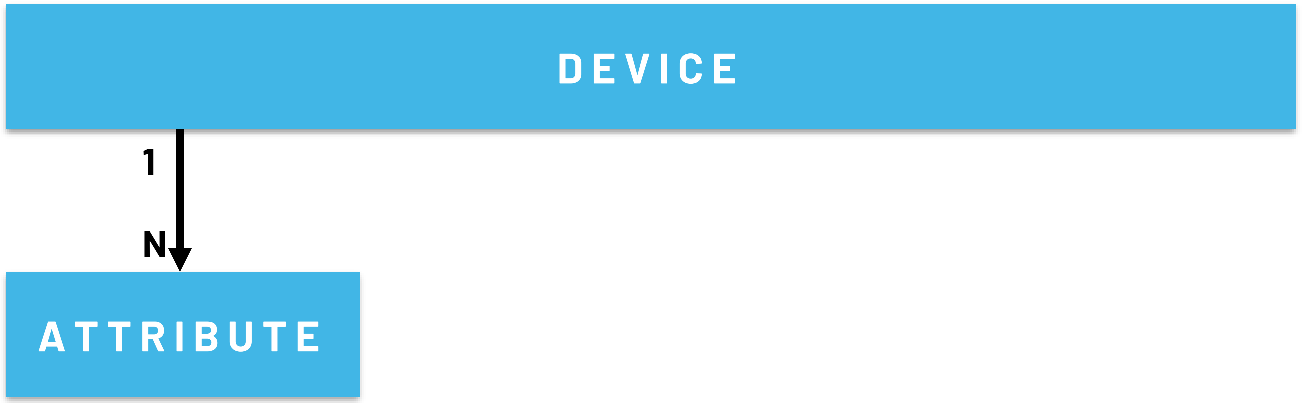
- Main structure
- Typically corresponds to a single physical hardware device
- Represented as directories in sysfs





IIO – Attributes

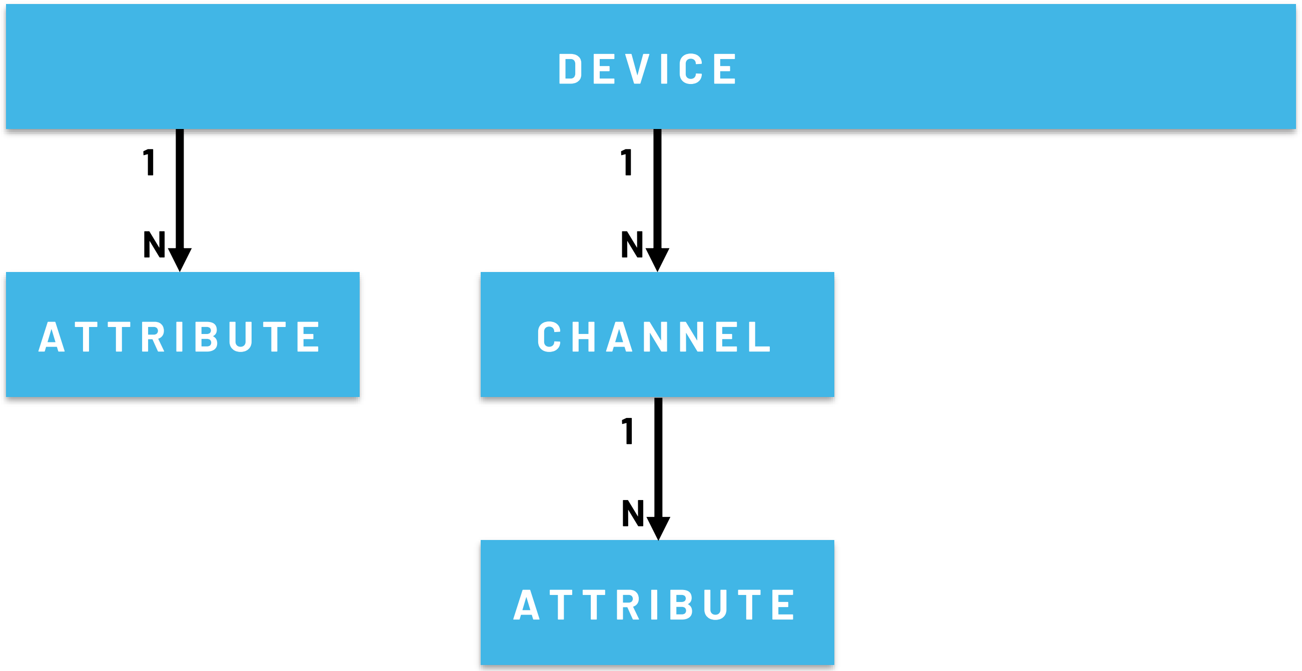
- Describe hardware capabilities
- Allow to configure hardware features
  - SAMPLING_FREQUENCY
  - POWERDOWN
  - PLL_LOCKED
  - SYNC_DIVIDERS
  - etc.
- Represented as files in sysfs



```
# ls /sys/bus/iio/devices/
iio:device0  iio:device1  iio:device2  iio:device3  iio:device4
# cat  /sys/bus/iio/devices/*/name
adm1177
ad9361-phy
xadc
cf-ad9361-dds-core-lpc
cf-ad9361-lpc
#
```

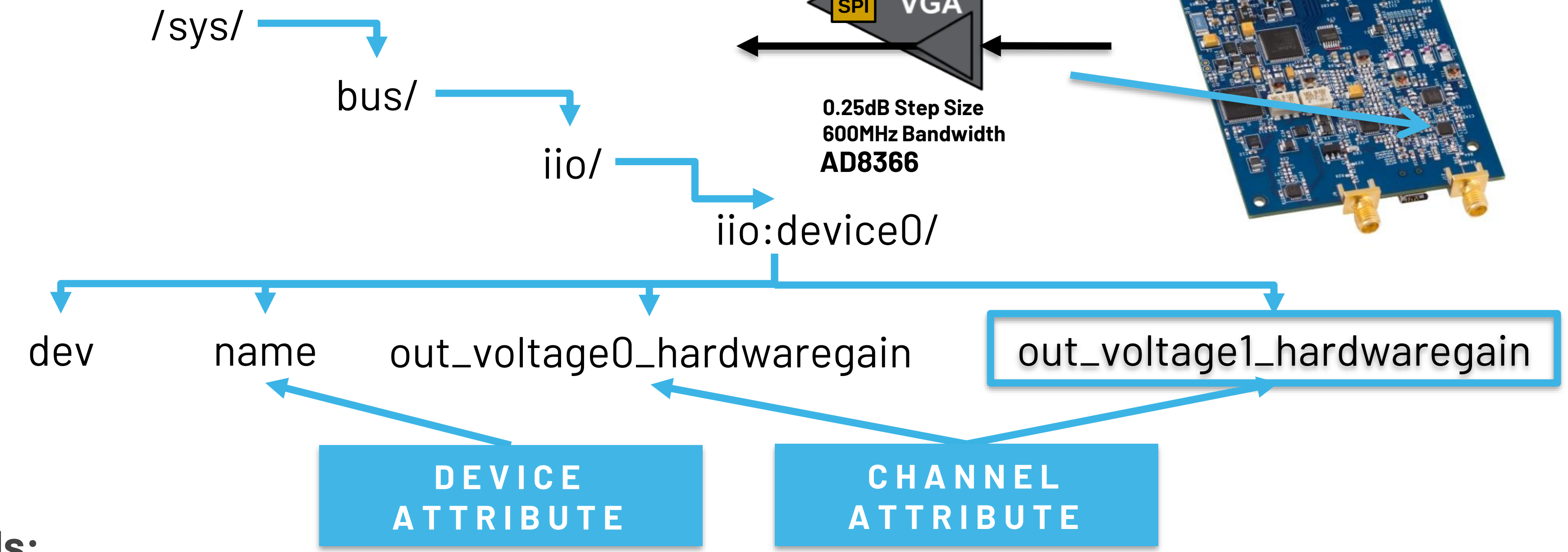
IIO – Channels

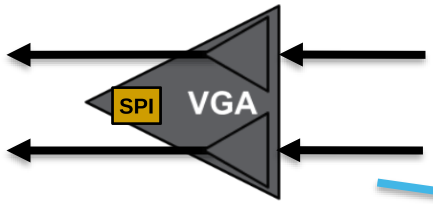
- Representation of a data channel
- Has direction, type, index and modifier
  - Direction
    - IN
    - OUT
  - Type
    - IIO_VOLTAGE
    - IIO_TEMP, etc.
  - Index
    - 0..N
  - Modifier
    - IIO_MOD_I, IIO_MOD_Q
- Channel Attributes provide additional information
  - RAW
  - SCALE
  - OFFSET
  - FREQUENCY
  - PHASE
  - HARDWAREGAIN
  - etc.



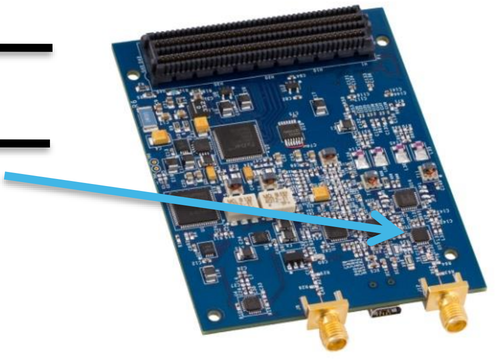
- Example: Read voltage from ADC Channel X in mV
  - *VoltageX_mV = (in_voltageX_raw + in_voltageX_offset) * in_voltageX_scale*

Example Device: AD8366 VGA/PGA Gain Control





0.25dB Step Size
600MHz Bandwidth
AD8366

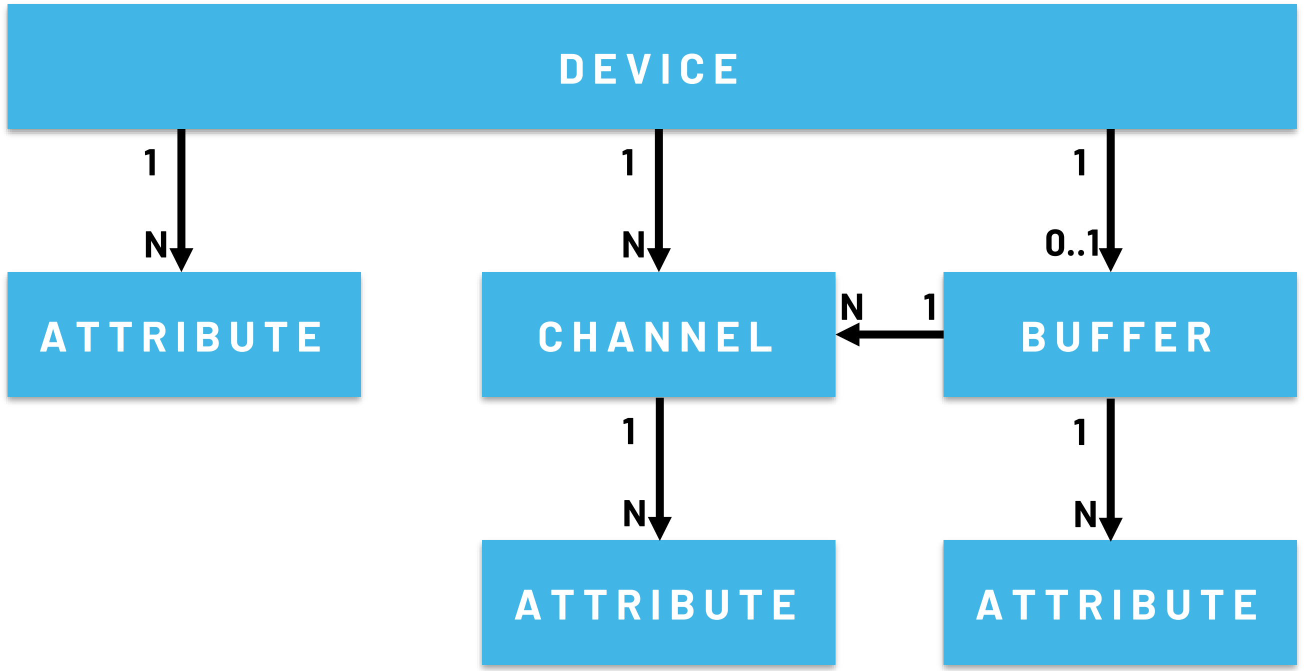


Shell Commands:

```
/sys/bus/iio/iio:device0 # cat name
ad8366-lpc
/sys/bus/iio/iio:device0 # echo 6 > out_voltage1_hardwaregain
/sys/bus/iio/iio:device0 # cat out_voltage1_hardwaregain
5.765000 dB
```

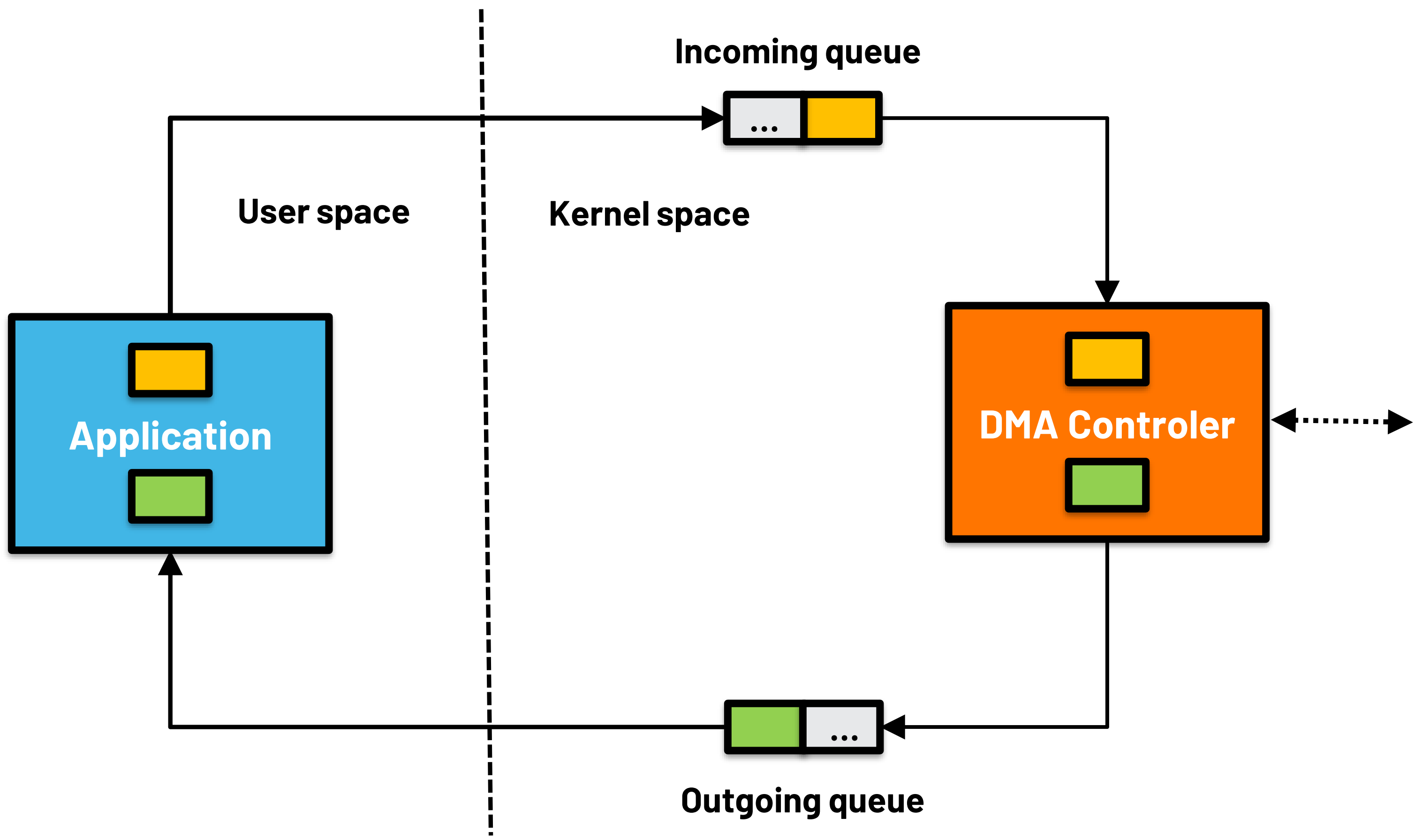
IIO – Buffers

- Used for continuous data capture/transmit
- Channels can be enabled/disabled
- Channels specify their data layout
  - [be|le]:[s|u]bits/storagebitsXrepeat[>>shift]
- /dev/iio:deviceX allows read()/write() access
- Configuration using sysfs files
- Support for different buffer implementations
  - Software FIFO
  - DMA Buffer
  - Device specific buffer



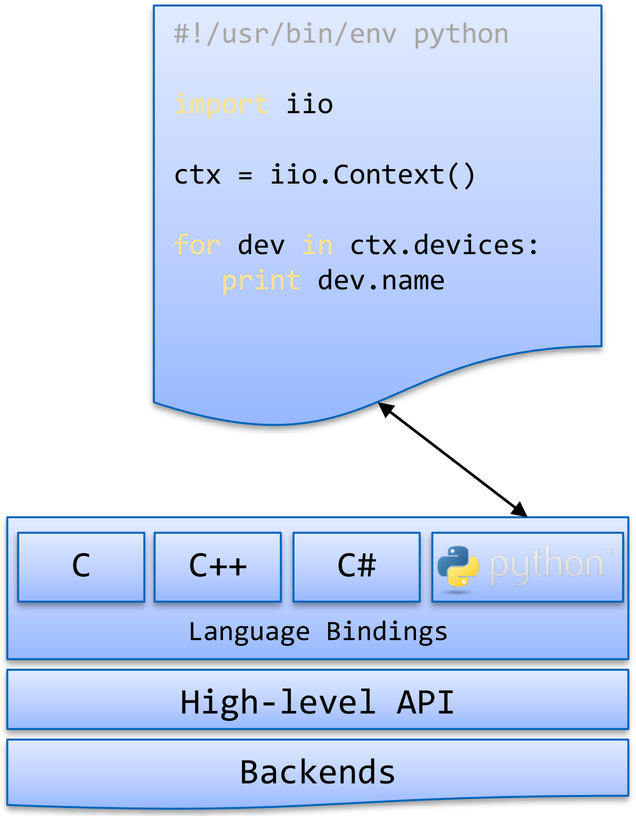
IIO – DMA buffer

- DMA is used to copy data from device to memory
- mmap() is used to make data available in the application
- Allows low overhead high-speed data capture
- Data is grouped into chunks (called DMA blocks) to manage ownership
  - Either application or driver/hardware owns a block
  - Samples per block are configurable
  - Number of blocks are configurable



IIO – libiio

- System library
- Abstracts away low level details of the IIO kernel ABI
  - Kernel ABI is designed to be simple and efficient
  - libiio focuses on ease of use
- Provides high-level C, C++, C# or Python programming interface to IIO (Language bindings)
  - Write your IIO application in your favorite language
- Cross Platform (Linux, Windows, MacOS X, BSD)
- Available as
  - Official DEBIAN package
  - RPM package
  - OpenEmbedded Layer meta-oe/libiio
  - Buildtroot package
  - Windows or Mac OS X installer
  - Etc.



```
#!/usr/bin/env python

import iio

ctx = iio.Context()

for dev in ctx.devices:
   print dev.name
```

For more information:

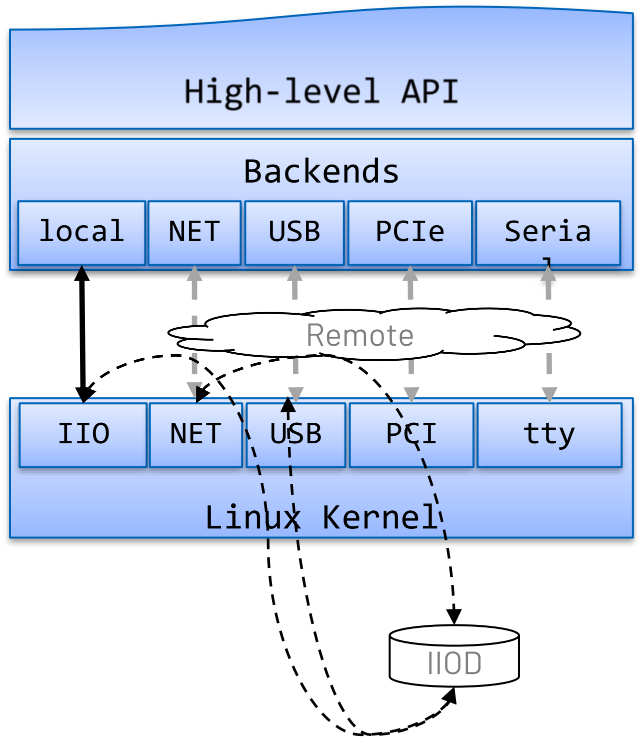
https://github.com/analogdevicesinc/libiio

http://wiki.analog.com/resources/tools-software/linux-software/libiio_internals

http://analogdevicesinc.github.io/libiio/

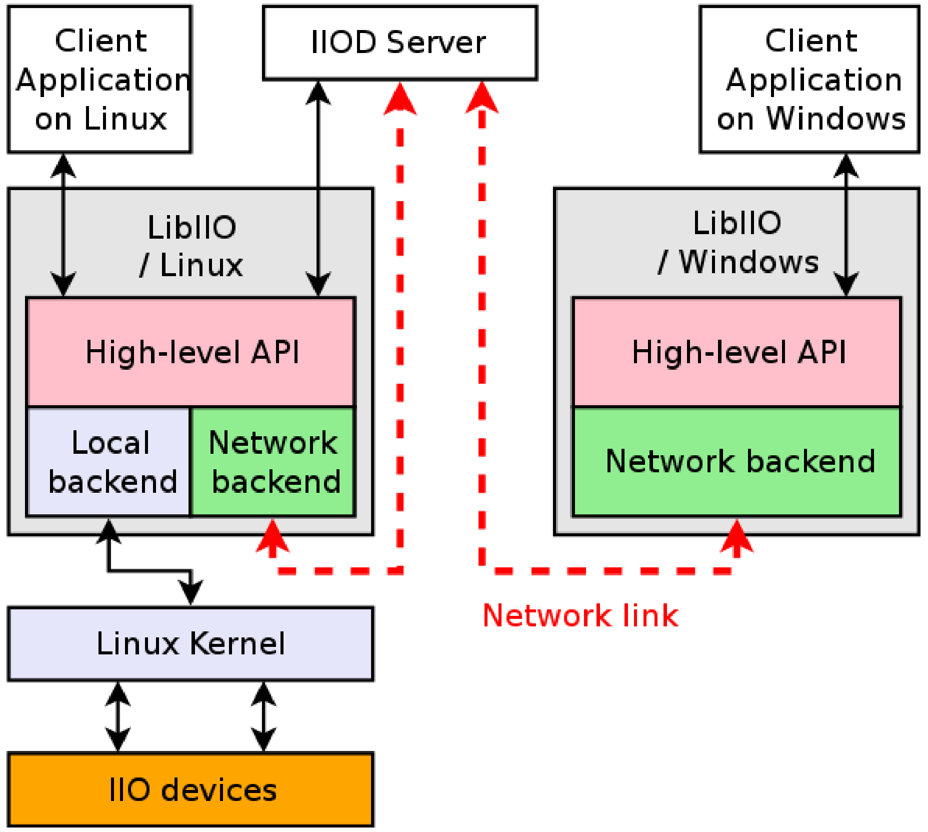
IIO – libiio – Backends

- Support for backends
  - Backend takes care of low-level communication details
  - Provide the same API for applications
  - Transparent from the applications point of view
- Multiple backends
  - Local, directly uses the Linux kernel IIO ABI
  - Network, uses network protocol to talk to (remote) iiod server which uses it's local backend
  - USB, SERIAL
- Allows to create flexible and portable applications
  - Write once, deploy everywhere
  - E.g. develop application on PC, deploy on embedded system (SoC, FPGA)



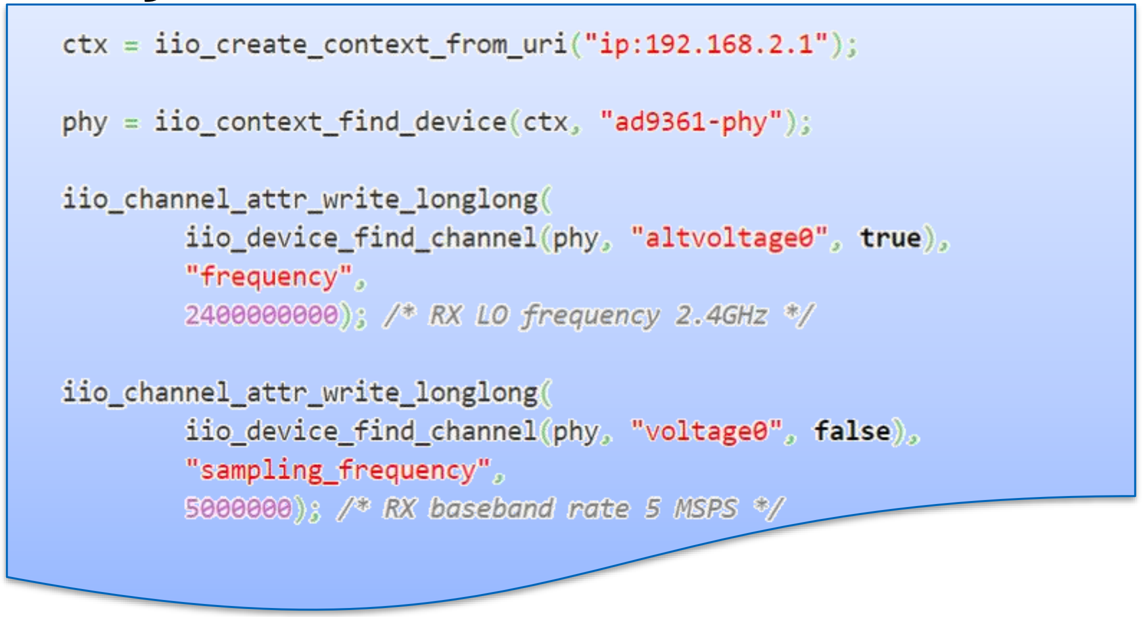
IIO – iiod

- Allows multiplexing between multiple readers/writers
- Provides support for remote clients via:
  - TCP/IP
  - USB
  - Serial
- Applications do not need system level privileges
- Transparent from the applications point of view



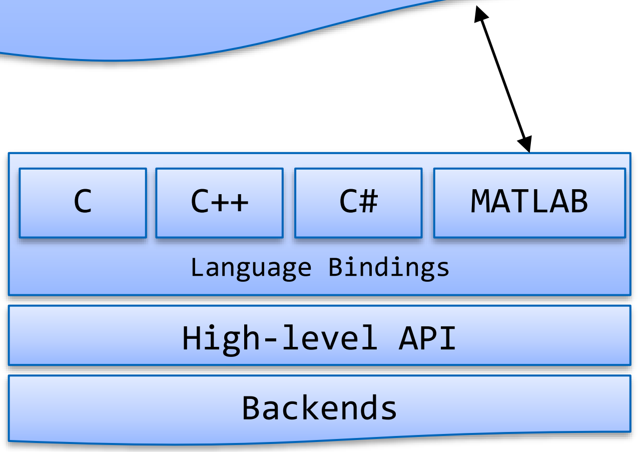
IIO – libiio

- System library
- Abstracts away low level details of the IIO kernel ABI
  - Kernel ABI is designed to be simple and efficient
  - libiio focuses on ease of use
- Provides high-level C, C++, C# or Python programming interface to IIO (Language bindings)



```
ctx = iio_create_context_from_uri("ip:192.168.2.1");

phy = iio_context_find_device(ctx, "ad9361-phy");

iio_channel_attr_write_longlong(
        iio_device_find_channel(phy, "altvoltage0", true),
        "frequency",
        2400000000); /* RX LO frequency 2.4GHz */

iio_channel_attr_write_longlong(
        iio_device_find_channel(phy, "voltage0", false),
        "sampling_frequency",
        5000000); /* RX baseband rate 5 MSPS */
```

```
rx = adi.AD9361.Rx('uri','ip:192.168.2.1’);
rx.channelCount = 2;
rx.CenterFrequency = 2.4e9;
rx.SamplingRate = 5e6;

for k=1:10
    valid = false;
    while ~valid
            [out, valid] = rx();
     end
end
```



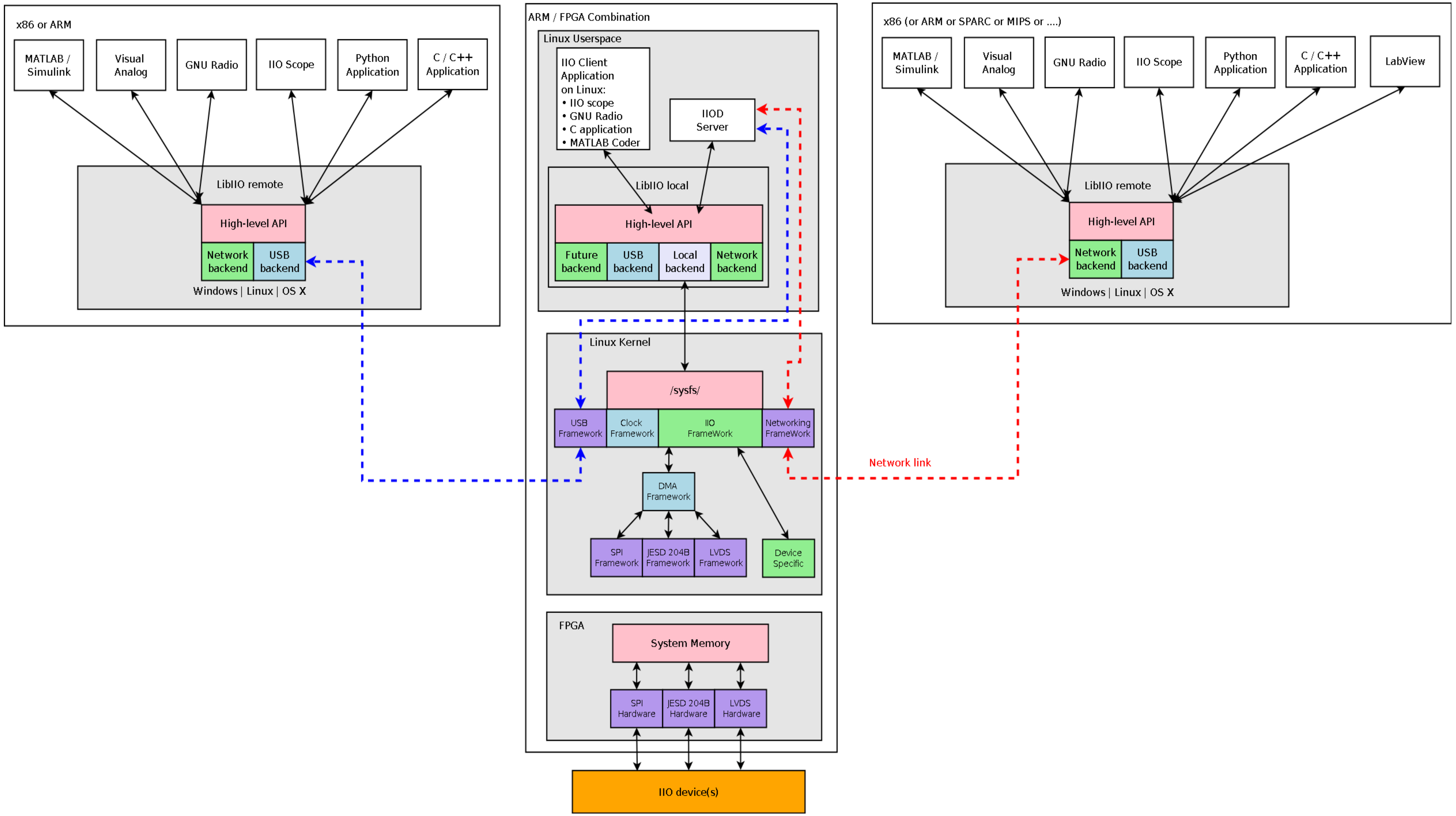
For more information:

https://github.com/analogdevicesinc/libiio

http://wiki.analog.com/resources/tools-software/linux-software/libiio_internals

http://analogdevicesinc.github.io/libiio/

IIO – Architecture Summarized



Infrastructure

Standardize Stack and Component Integration

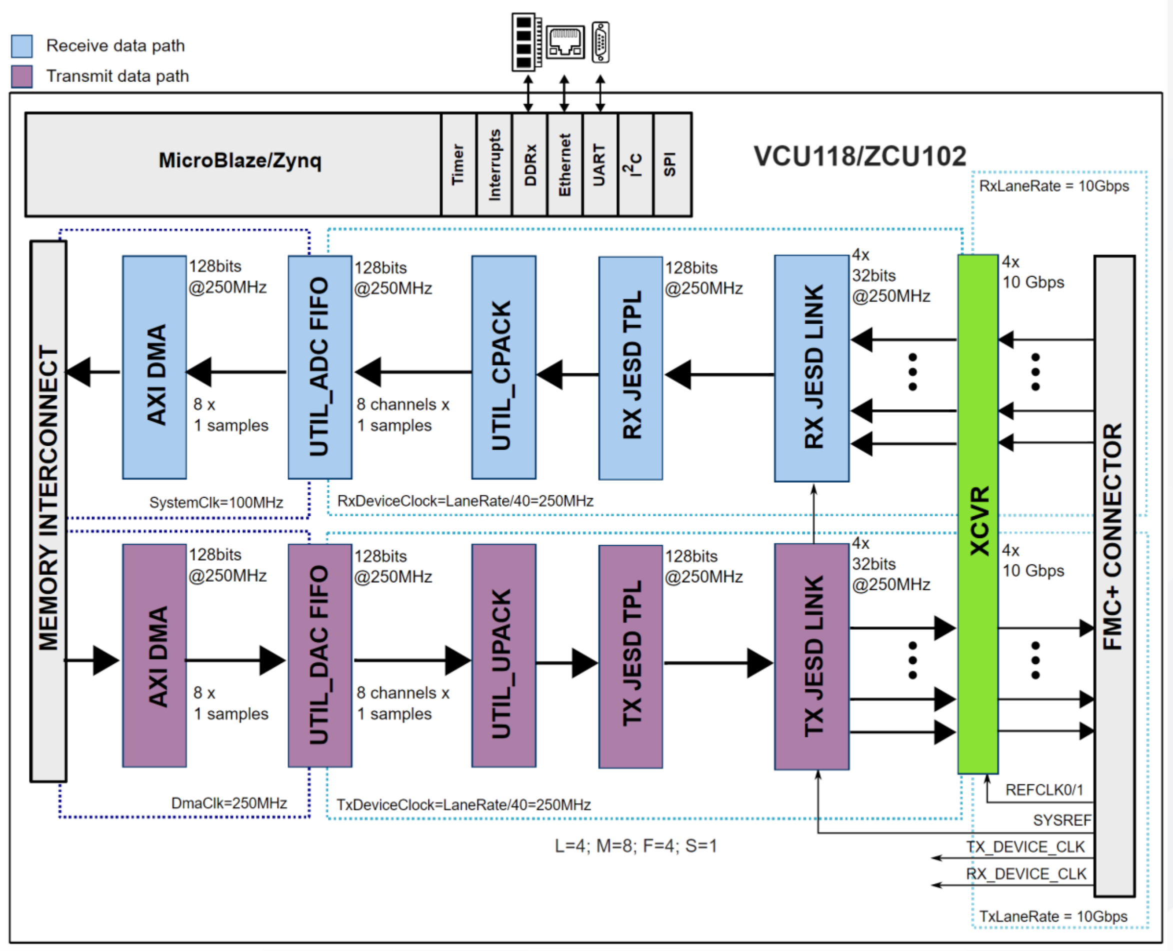
- Key Concepts:
  - All components are open, supported, and documented (Not all by ADI – Industry Standards)
  - Customers can take ***all*** or pieces of complete stack for end-product
  - All components have a long maintenance model
  - Released components have long history +10 years (customers already familiar)
- Design Strategy
  - Provide starting point close to customer end system
  - Maximal reuse of components
  - Minimize learning between platforms
  - Justification through customer request and ROI
  - Be aware of ecosystem requirements
  - Leverage modern practices but minimize disruptive changes
  - Be transparent
  - Optimize for customer systems not internal development

- Supporting many customer system level use cases
- Providing tools for debug
- Meeting customer at their tools

Full Stack (IIO/Linux based) HDL Designs

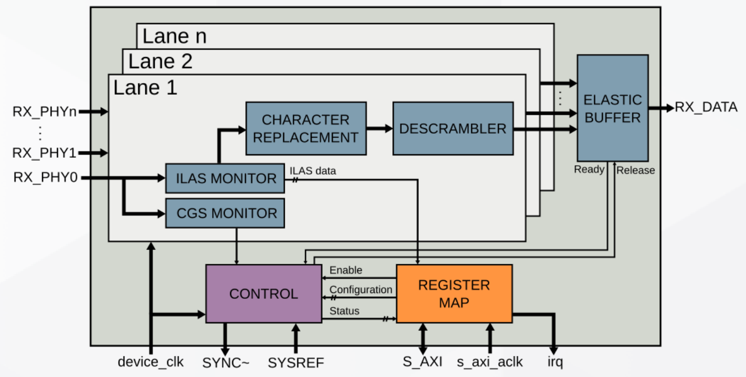
- Common HDL across all IIO reference designs
- Full stack reference Designs include JESD204 physical layers (XCVR); link layers; and transport layers.
  - termination to DMA via a AXI-STREAM or FIFO interface
  - Runs across different Intel and Xilinx carriers
- Are designed to be disconnected to "insert custom signal processing"
  - Your modem, Your signals intelligence, etc
- Example designs show how to use MUX in different places in design, to stream debug data (I/Q samples, or payload (data buffers)) as IIO streams.
- Works with industry standard debug tools:
  - Xilinx Integrated Logic Analyzer
  - Intel Signal Tap
  - MathWorks HDL Verifier (in SoC Blockset)

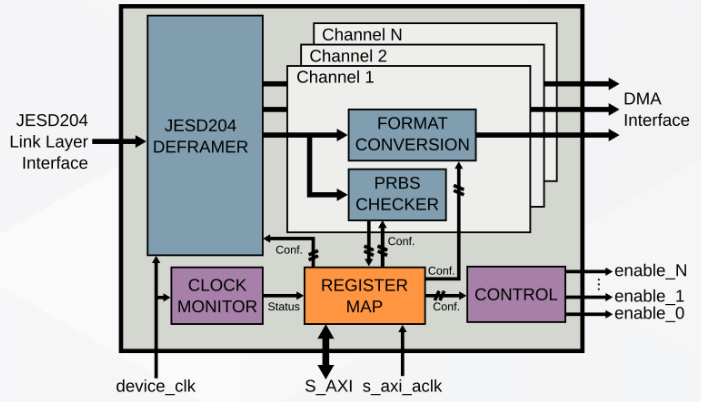
ample block design for Single Link; M=8; L=4;

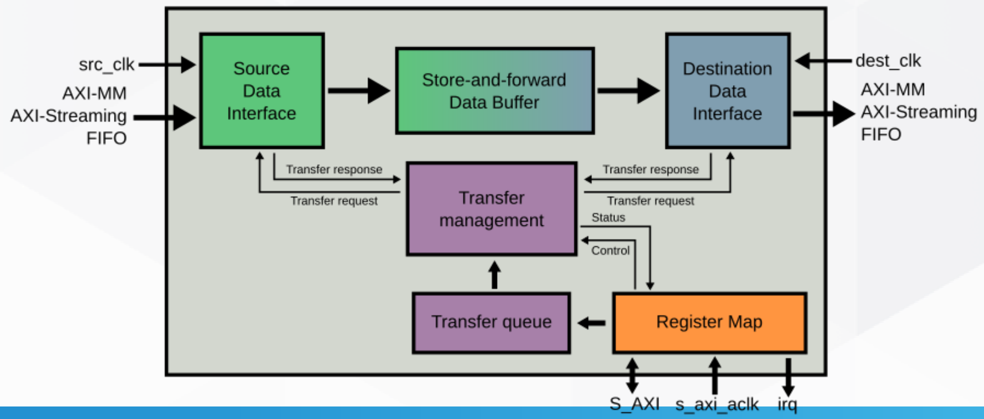


FPGA HDL Cores

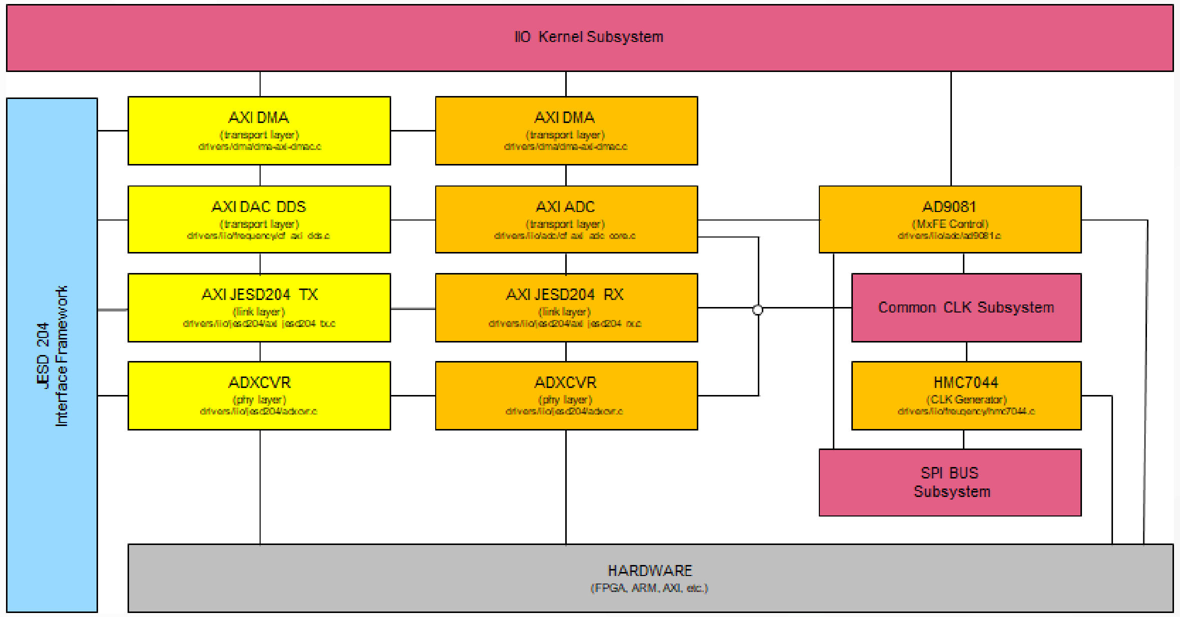
- We design and support interface logic for various ADI IC devices, encapsulating any and all obscure designs, to provide a **common, easy to understand** and **consistent interface** that is **portable** across **commonly available** FPGA devices and standard cell libraries.
  - Full stack IP cores to manage physical, link and transport layers, using standard interfaces. (AXI, FIFO)
  - Portable IP cores for Intel/Altera Quartus and Xilinx Vivado
  - Open source, documentation and support www.github.com/analogdevicesinc/hdl
  - Verilog HDL source files, Vivado and Intel constraint files
  - Updated and maintained to the latest versions of the tools
  - Tested and proven on various hardware platforms
  - Modular design, easy to use, scale, debug and customize



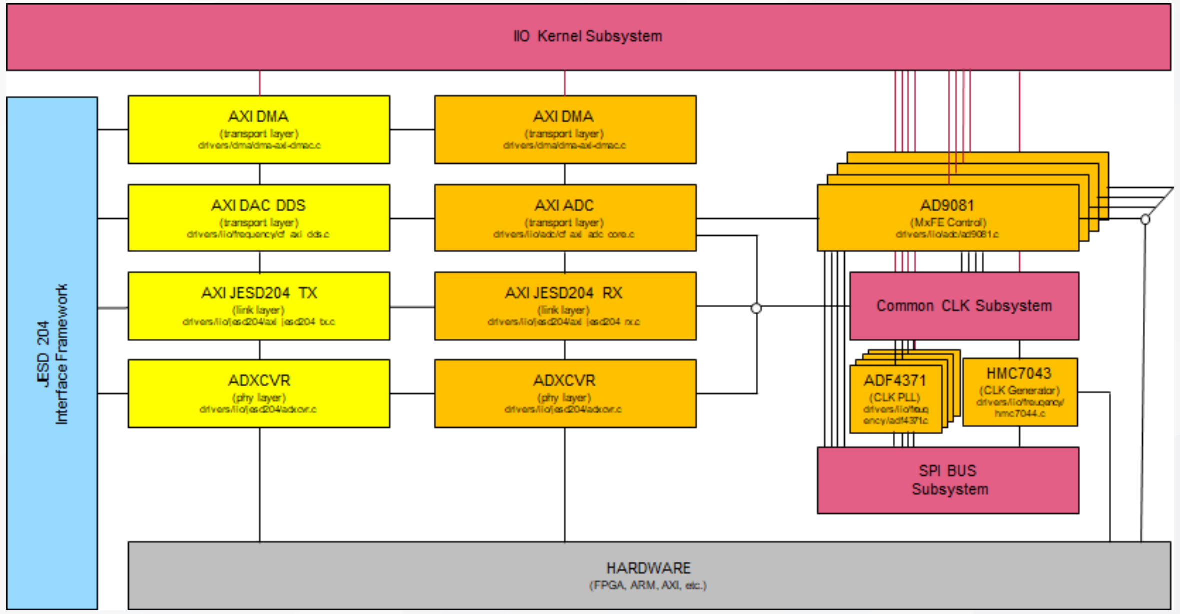




Modular Stack: Following the Industry


IIO Kernel Subsystem
JESD 204 Interface Framework
AXI DMA (transport layer) drivers/dma/dma-axi-dmac.c
AXI DMA (transport layer) drivers/dma/dma-axi-dmac.c
AXI DAC DDS (transport layer) drivers/iio/frequency/cf_axi_dds.c
AXI ADC (transport layer) drivers/iio/adc/cf_axi_adc_core.c
AXI JESD204 TX (link layer) drivers/iio/jesd204/axi_jesd204_tx.c
AXI JESD204 RX (link layer) drivers/iio/jesd204/axi_jesd204_rx.c
ADXCVR (phy layer) drivers/iio/jesd204/adxcvr.c
ADXCVR (phy layer) drivers/iio/jesd204/adxcvr.c
AD9081 (MxFE Control) drivers/iio/adc/ad9081.c
Common CLK Subsystem
HMC7044 (CLK Generator) drivers/iio/frequency/hmc7044.c
SPI BUS Subsystem
HARDWARE (FPGA, ARM, AXI, etc.)

AD9081-FMCA-EBZ

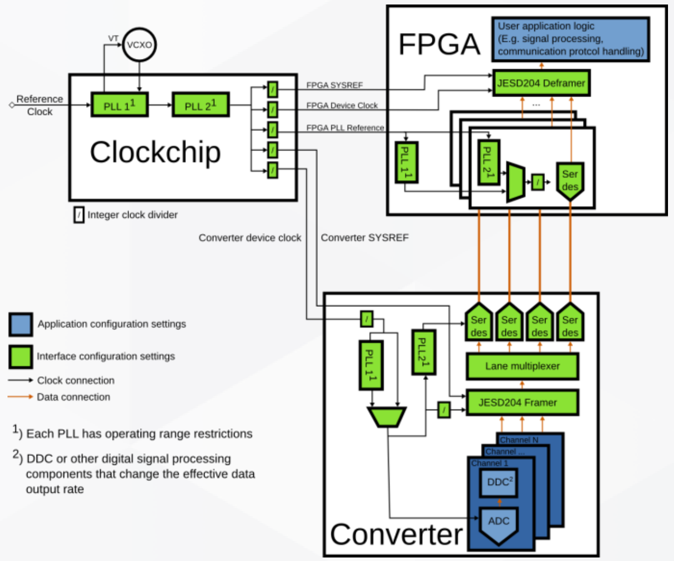

IIO Kernel Subsystem
JESD 204 Interface Framework
AXI DMA (transport layer) drivers/dma/dma-axi-dmac.c
AXI DMA (transport layer) drivers/dma/dma-axi-dmac.c
AXI DAC DDS (transport layer) drivers/iio/frequency/cf_axi_dds.c
AXI ADC (transport layer) drivers/iio/adc/cf_axi_adc_core.c
AXI JESD204 TX (link layer) drivers/iio/jesd204/axi_jesd204_tx.c
AXI JESD204 RX (link layer) drivers/iio/jesd204/axi_jesd204_rx.c
ADXCVR (phy layer) drivers/iio/jesd204/adxcvr.c
ADXCVR (phy layer) drivers/iio/jesd204/adxcvr.c
AD9081 (MxFE Control) drivers/iio/adc/ad9081.c
Common CLK Subsystem
ADF4371 (CLK PLL) drivers/iio/frequency/adf4371.c
HMC7043 (CLK Generator) drivers/iio/frequency/hmc7044.c
SPI BUS Subsystem
HARDWARE (FPGA, ARM, AXI, etc.)

ADQUADMXFE1EBZ

JESD204 Interface Framework

- System-level integrated HDL and software framework covering the whole stack
  - **Hardware**: Reference and rapid prototyping systems
  - **HDL**: Components for JESD204 protocol handling
  - **Software**: Drivers to manage clock-chips, converters and HDL
- Components have been co-designed for improved interoperability
- Key features
  - Automatic interface configuration based on application settings
    - High-level API
  - Dynamic re-configuration
  - Improved diagnostics
- ADI provides full stack reference designs
  - Works out of the box
  - Starting point for development of custom designs

JESD204
Interface Framework



Application	Application specific processing
Transport	Sample framing Lane mapping
Link	Scrambling Character replacement 8b/10b encoding
Physical	High-speed SerDes Clock recovery Signal shaping

Third Party Tool Integration
Matlab/Simulink Python
GNU Radio

Software Reference Designs
Configures drivers for HDL and hardware reference design

Software System Libraries
Provides unified interface (API)
Implements common system tasks

Software Drivers
Manages Hardware Components
Manages HDL Components

HDL Reference Designs
Instanciates HDL IP for hardware reference design

HDL IP Components
Phy Layer Transport Layer
Link Layer DMA

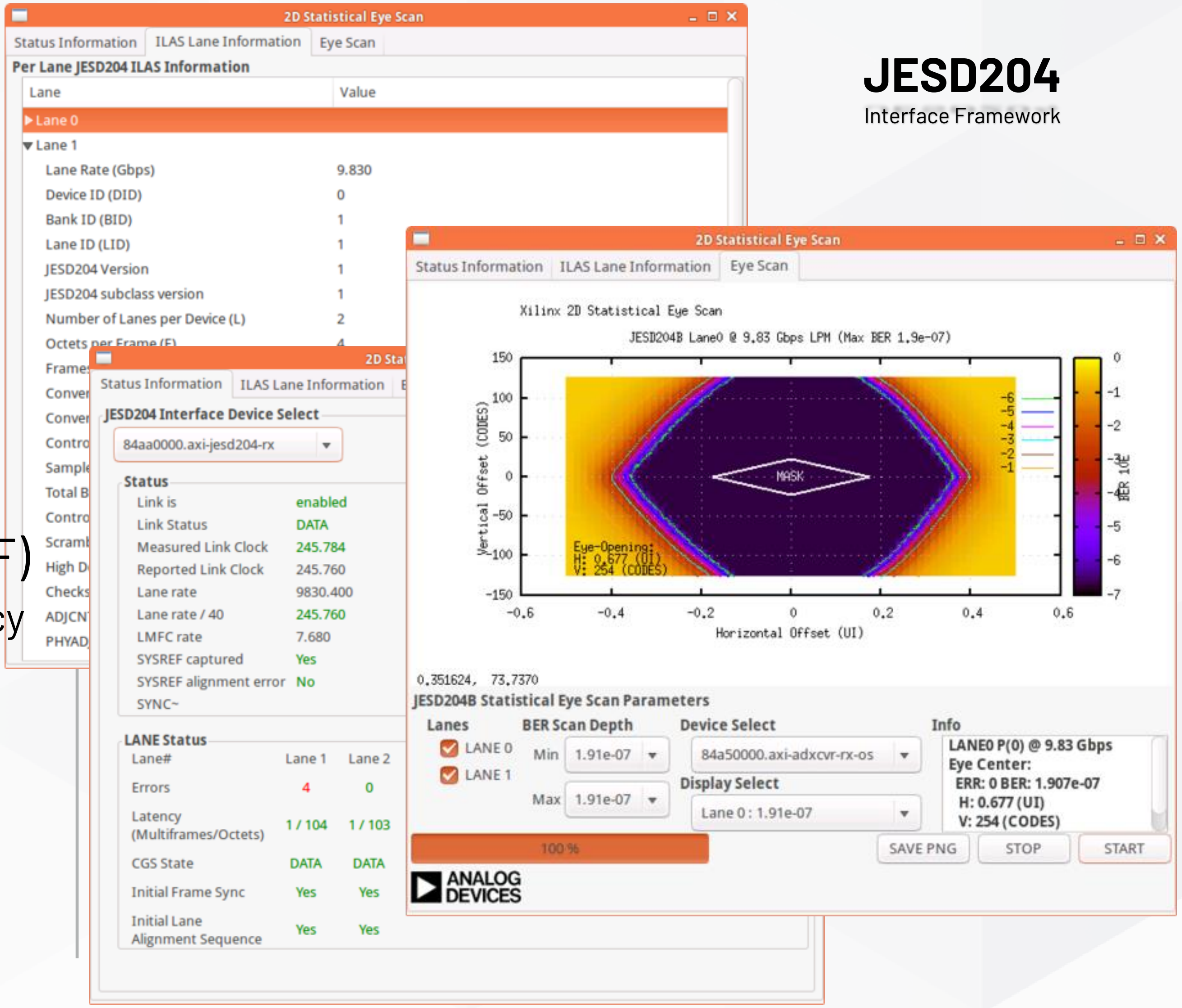
Hardware Reference Desgins
Rapid prototyping boards
Evaluation boards

Hardware Components
Converter Power
Clockchip Analog Frontend

Diagnostics (debugability)

- Statistical Eyescan, Bit-error-rate monitoring
  - Detect electrical signal integrity issues
- Clock rate monitoring for all system clocks
  - Detect bad clock wiring
  - Detect clock failures
- Initial lane sequence monitoring and verification
  - Detect lane swaps
- Lane arrival monitoring (relative to SYSREF)
  - Detect potential sources of non-deterministic latency
- SYSREF alignment monitoring
  - Detect SYSREF timing and configuration issues
- Continuous monitoring
  - Application is notified as soon as failure occurs

JESD204
Interface Framework



2D Statistical Eye Scan

Status Information | ILAS Lane Information | Eye Scan

Per Lane JESD204 ILAS Information

Lane	Value
▶ Lane 0	
▼ Lane 1	
Lane Rate (Gbps)	9.830
Device ID (DID)	0
Bank ID (BID)	1
Lane ID (LID)	1
JESD204 Version	1
JESD204 subclass version	1
Number of Lanes per Device (L)	2
Octets per Frame (F)	4
Frame	
Conver	
Conver	
Contro	
Sample	
Total B	
Contro	
Scraml	
High D	
Checks	
ADJCN	
PHYAD	

2D Sta

Status Information | ILAS Lane Information | E

JESD204 Interface Device Select

84aa0000.axi-jesd204-rx

Status

Link is	enabled
Link Status	DATA
Measured Link Clock	245.784
Reported Link Clock	245.760
Lane rate	9830.400
Lane rate / 40	245.760
LMFC rate	7.680
SYSREF captured	Yes
SYSREF alignment error	No
SYNC~	

LANE Status

Lane#	Lane 1	Lane 2
Errors	4	0
Latency (Multiframes/Octets)	1 / 104	1 / 103
CGS State	DATA	DATA
Initial Frame Sync	Yes	Yes
Initial Lane Alignment Sequence	Yes	Yes

0.351624, 73.7370

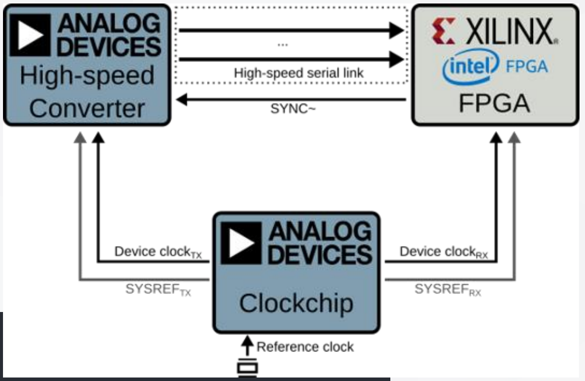
JESD204B Statistical Eye Scan Parameters

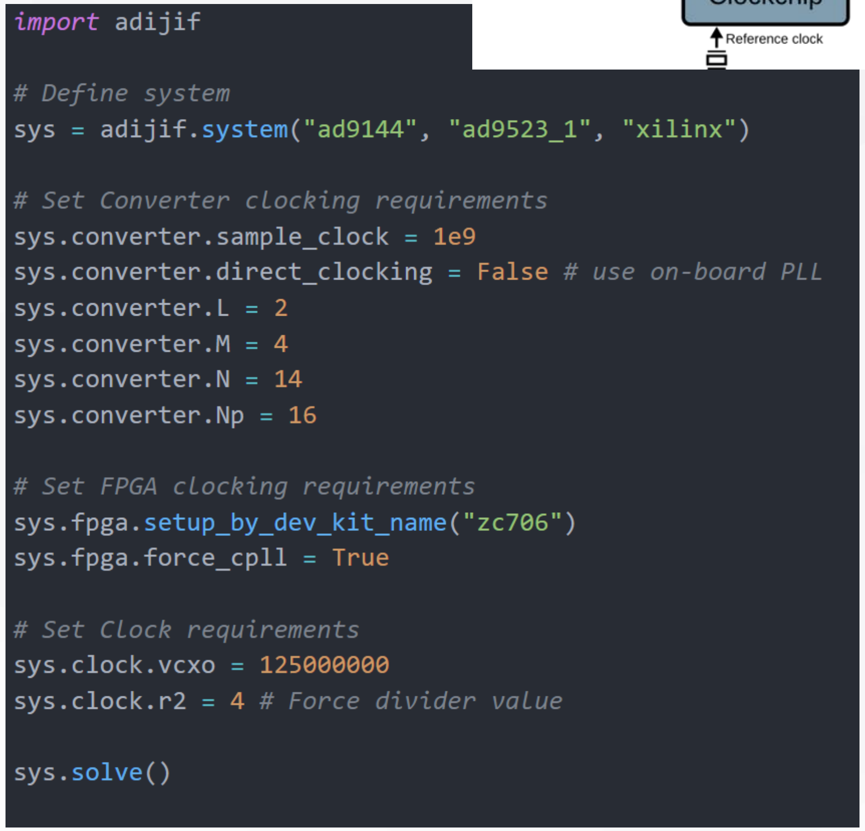
Lanes	BER Scan Depth	Device Select	Info
✓ LANE 0	Min 1.91e-07	84a50000.axi-adxcvr-rx-os	LANE0 P(0) @ 9.83 Gbps
✓ LANE 1	Max 1.91e-07	Display Select: Lane 0 : 1.91e-07	Eye Center: ERR: 0 BER: 1.907e-07 H: 0.677 (UI) V: 254 (CODES)

PyADI-JIF: JESD Interface Framework Configurator

- JESD204 and clock configuration solver and exploration tool
- Compliments ADI JESD Framework drivers and HDL IP
- Models converters, clock chips, and FPGAs together so all constraints are considered
- Supported parts include:
  - AD9680, AD9144, ADRV9009, AD9081/2, …
  - AD9523-1, AD9528, HMC7044, LTC6952, …
  - Xilinx FPGAs
  - More being added every day

PyADI-JIF
JESD204 FRAMEWORK CONFIGURATOR

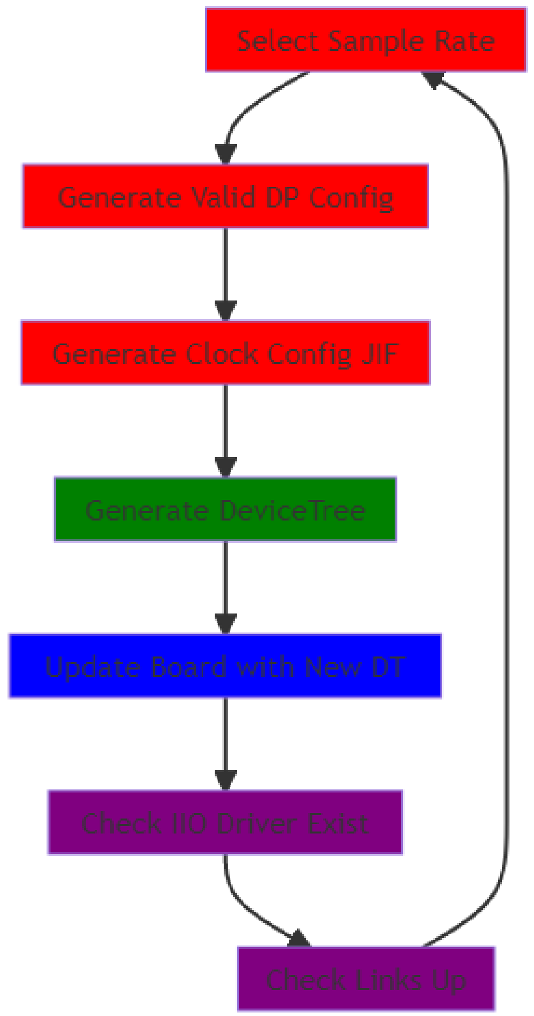



```python
import adijif

# Define system
sys = adijif.system("ad9144", "ad9523_1", "xilinx")

# Set Converter clocking requirements
sys.converter.sample_clock = 1e9
sys.converter.direct_clocking = False # use on-board PLL
sys.converter.L = 2
sys.converter.M = 4
sys.converter.N = 14
sys.converter.Np = 16

# Set FPGA clocking requirements
sys.fpga.setup_by_dev_kit_name("zc706")
sys.fpga.force_cpll = True

# Set Clock requirements
sys.clock.vcxo = 125000000
sys.clock.r2 = 4 # Force divider value

sys.solve()
```

JIF Workflow: High-Level Config To Hardware

End-To-End Test Flow

- JIF Workflow is a connected set of tests that show full end-to-end configuration to deployment to HW
- System level acceptance test

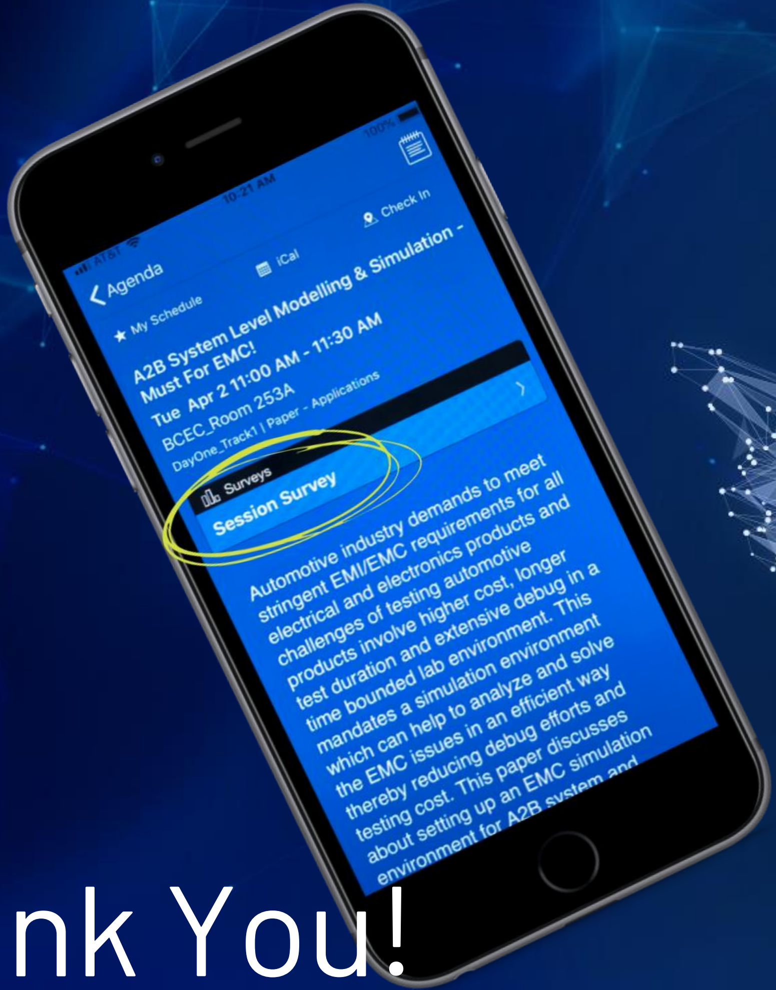


PyADI-JIF
JESD204 FRAMEWORK CONFIGURATOR

PyADI-DT
DEVICETREE MANAGEMENT UTILTIES

Nebula (optional)

PyADI-IIO
PYTHON FOR ADI INDUSTRIAL I/O DEVICES



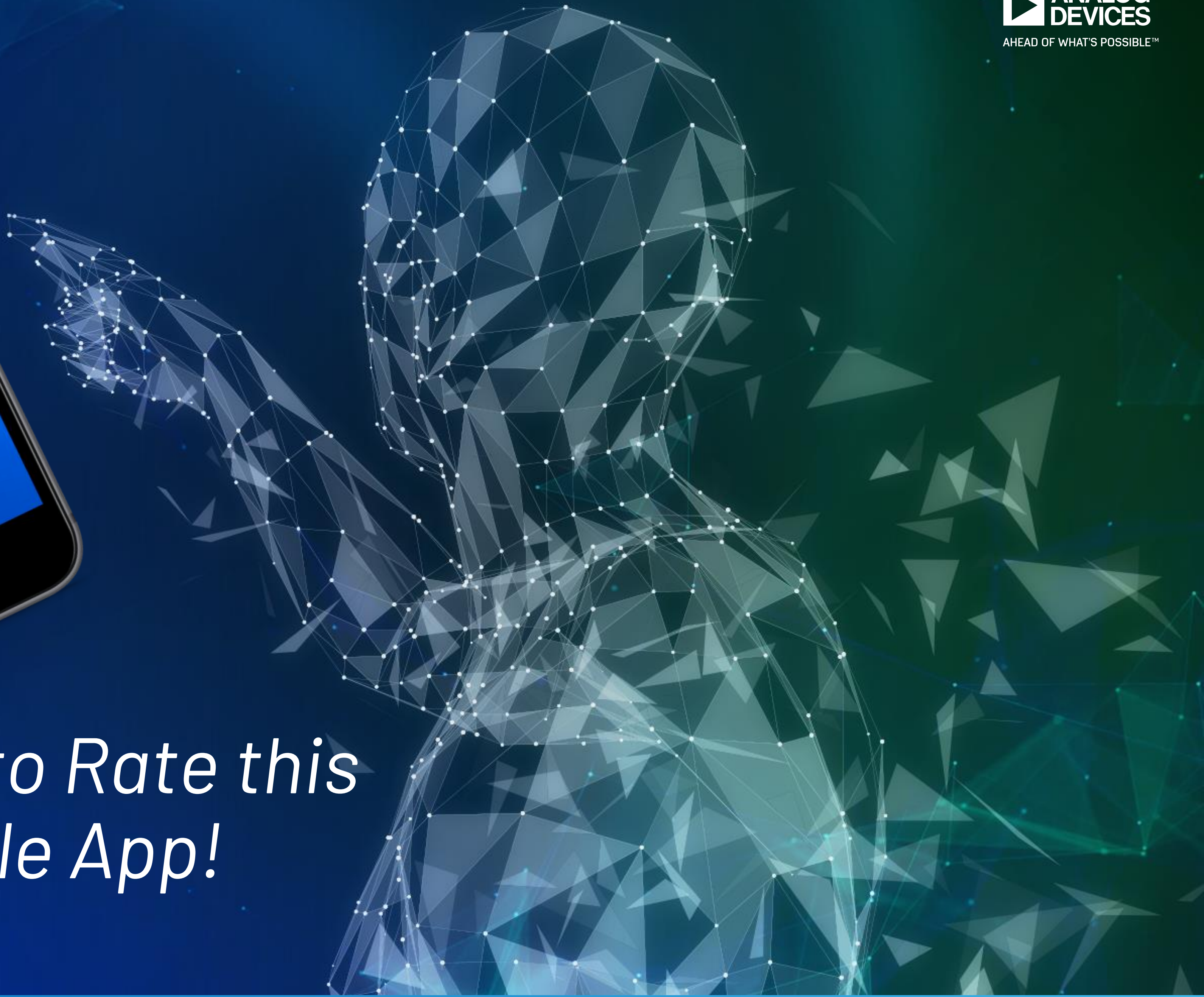
A2B System Level Modelling & Simulation - Must For EMC!

Tue Apr 2 11:00 AM - 11:30 AM

BCEC_Room 253A

DayOne_Track1 | Paper - Applications

Automotive industry demands to meet stringent EMI/EMC requirements for all electrical and electronics products and challenges of testing automotive products involve higher cost, longer test duration and extensive debug in a time bounded lab environment. This mandates a simulation environment which can help to analyze and solve the EMC issues in an efficient way thereby reducing debug efforts and testing cost. This paper discusses about setting up an EMC simulation environment for A2B system and



Thank You!

Please Remember to Rate this Session in the Mobile App!