

Rapid Phased Array Prototyping

CRAWL → WALK → RUN

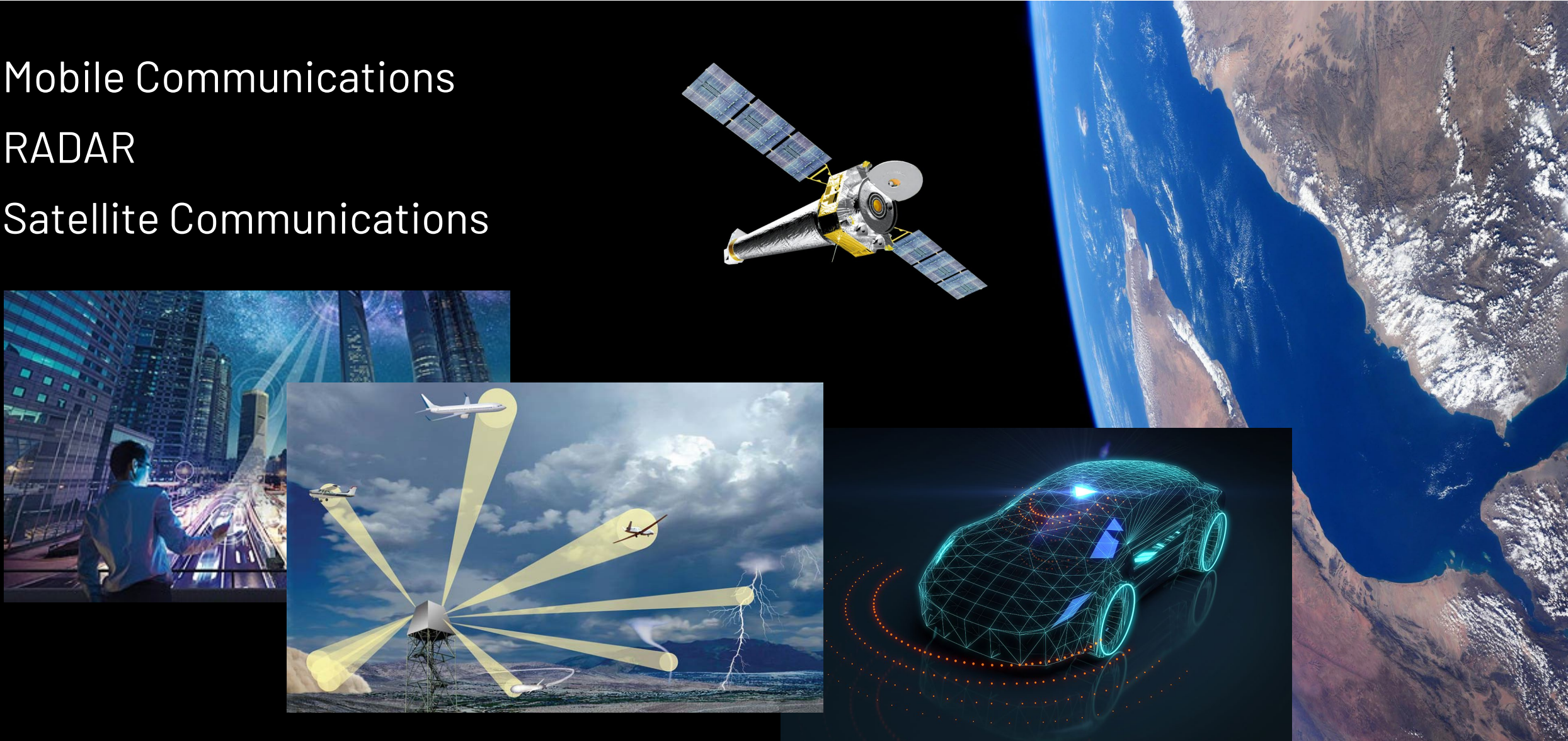
Jon Kraft, Analog Devices
June 12, 2023

Phased Array Antennas Are Being Used Everywhere

Mobile Communications

RADAR

Satellite Communications

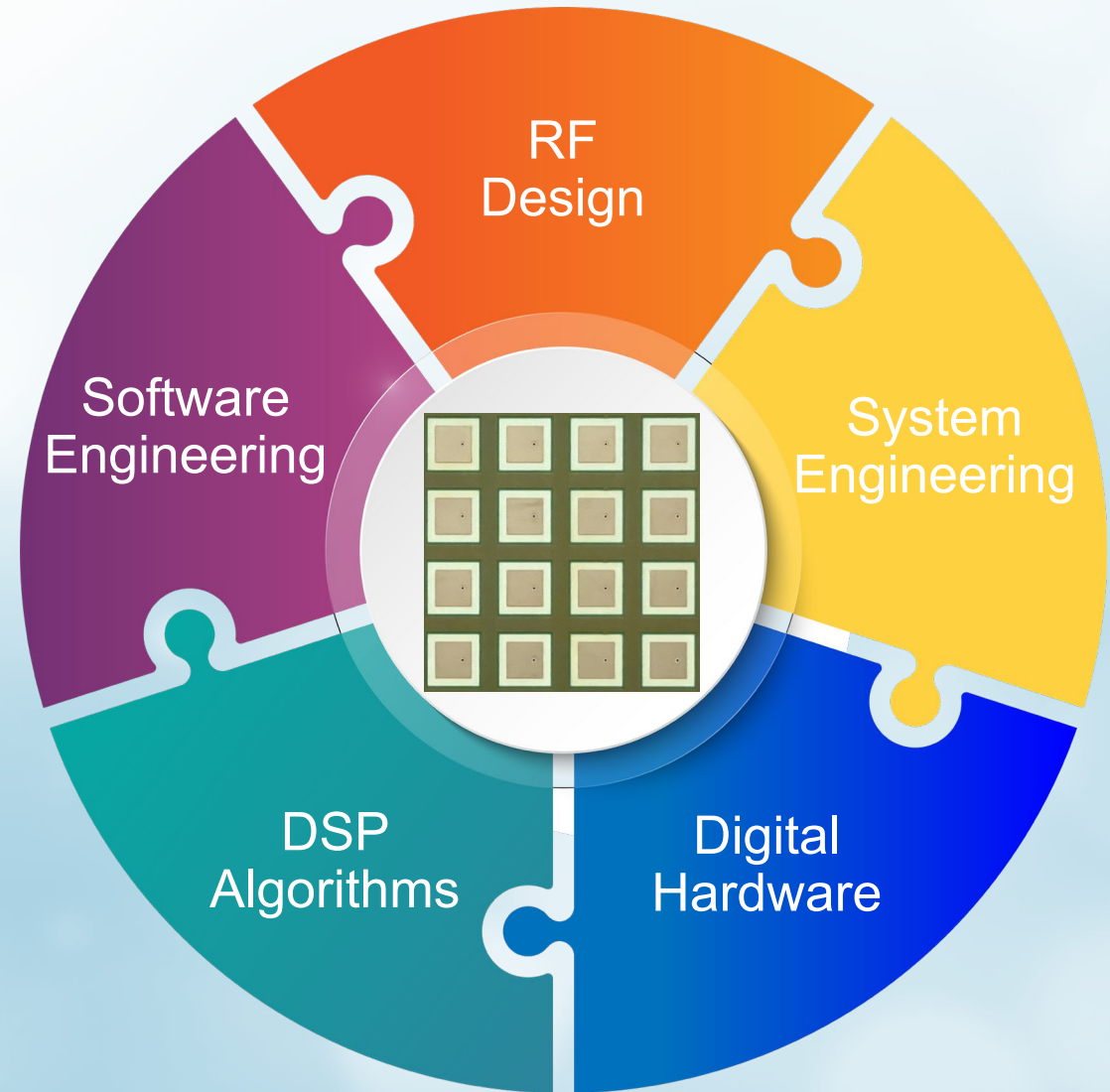


Challenges with Getting Started in Phased Arrays

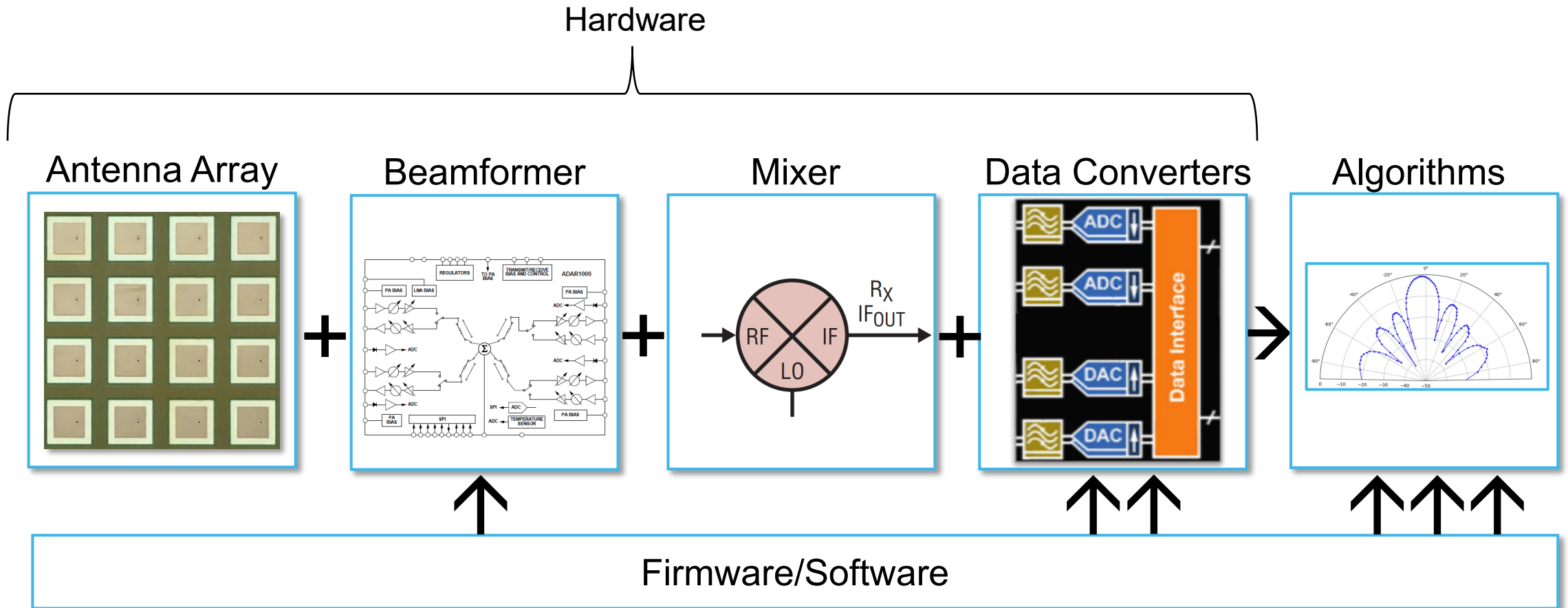
► Phased Array Radar Requires:

- RF Hardware Design
- Software Engineering
- System Design
- Algorithm Design (comms and radar)
- HDL Engineering

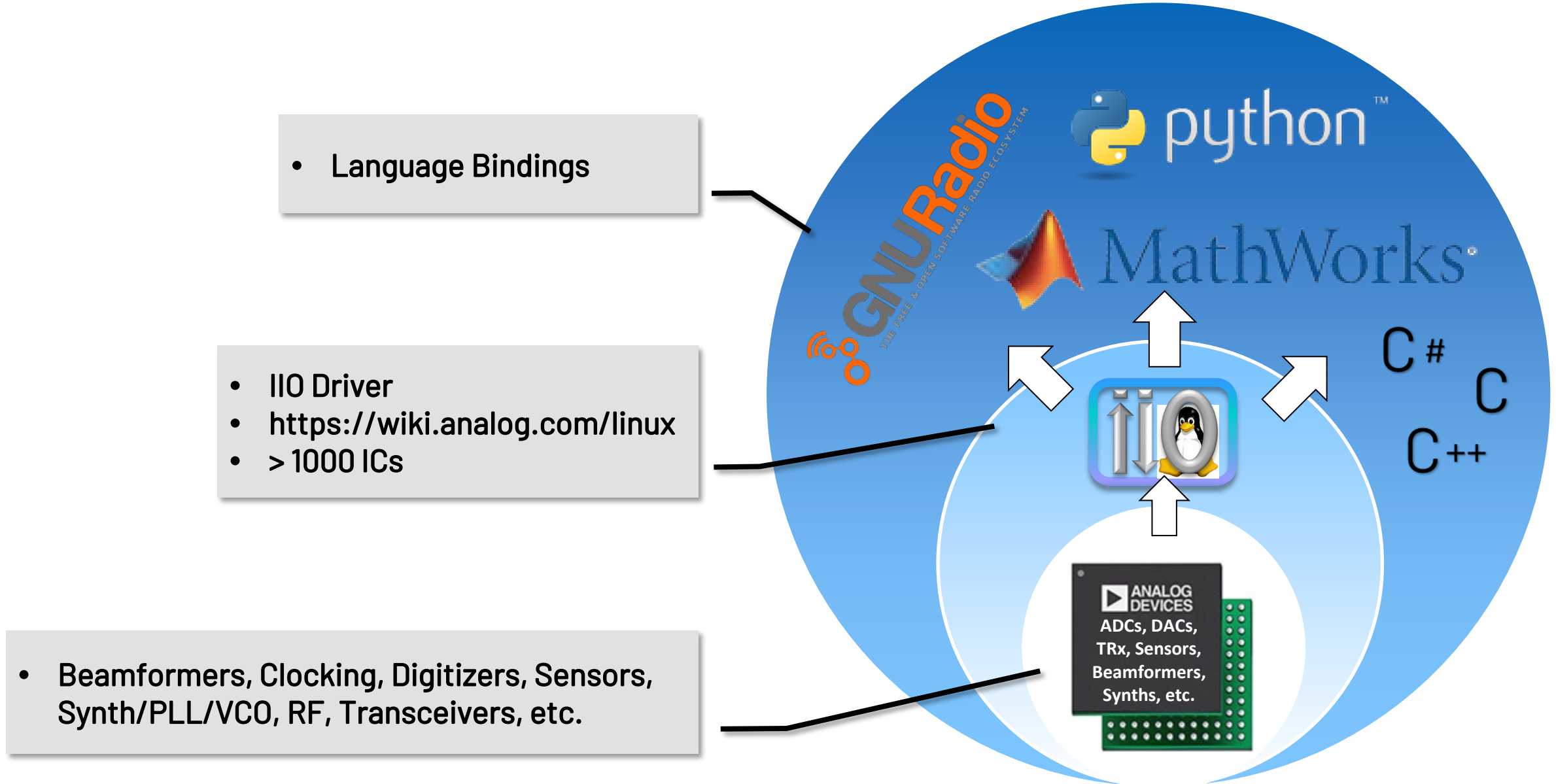
► So with so much entailed,
how can we get started?



A Simple Phased Array Radar



Common IIO Driver For All ICs and Environments



Contributors – Who is writing IIO Code?

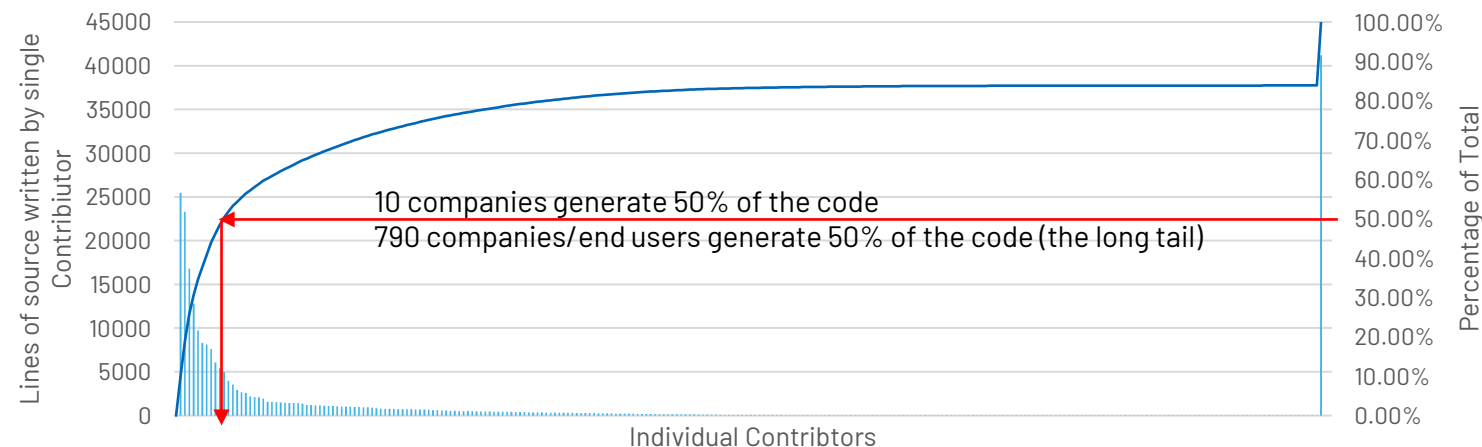
- Semiconductor Manufactures (40%), Contractors/Consultants/Tools Companies (10%), and End users (50%)

intel.com
analog.com
st.com
invensense.com
ti.com
microchip.com
freescale.com
atmel.com
nvidia.com
broadcom.com
renesas.com
rohmeurope.com
mediatek.com
maximintegrated.com
rock-chips.com
nxp.com
hisilicon.com
arm.com
xilinx.com
amlogic.com
mellanox.com
beagleboard.org
linux-mips.org
te.com
cirrus.com
idt.com
wolfsonmicro.com

linaro.org
redhat.com
pengutronix.de
free-electrons.com
bootlin.com
denx.de
iis.fraunhofer.de
linutronix.de
ibm.com
linux-foundation.org
canonical.com
synopsys.com
windriver.com
mathworks.com
suse.com

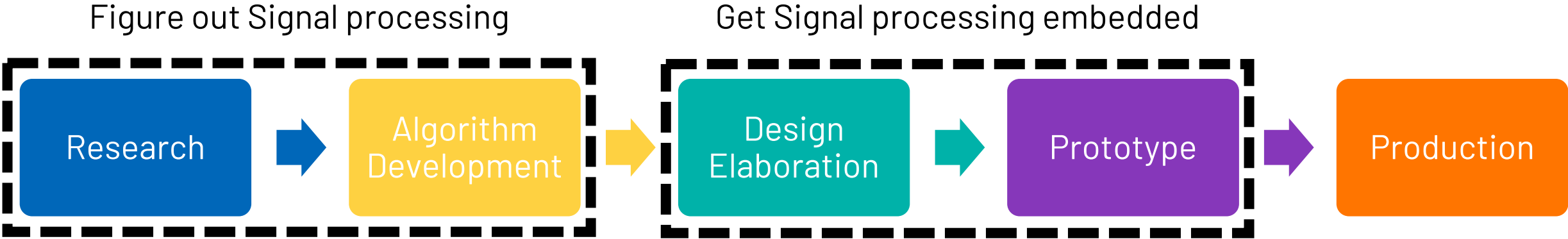
samsung.com
chromium.org
huawei.com
google.com
rockwellcollins.com
siemens.com
axis.com
ni.com
loewensteinmedical.de
rohde-schwarz.com
visteon.com
jdsu.com
cisco.com
sonymobile.com

Contributors "ownership"



Why start with IIO/Linux?

Not required to ship Linux to use IIO/Linux during development



- ▶ Use Linux, Don't become a Linux Developer
 - Some understanding is good, but don't need to write Linux drivers yourself.
- ▶ Networking, PCIe, 40G Ethernet, File Systems, USB, Connectivity, debuggability, all just work
- ▶ Standard frameworks for nearly every sort of device, JESD204, Clocks, SPI, etc
 - Vendor independence
- ▶ Powerful debug tools
- ▶ Take advantage of 15,600 developers from more than 1,400 companies
- ▶ Linux very popular (easy to ask questions, lots of books, lots of college interns)

Linux

Linux

or

No-OS / RTOS / API

- ▶ Use what you ship
- ▶ Optimize for your platform
- ▶ Not trying to make it function

► Linux:

- 1200+ in-kernel drivers (most upstreamed)
- High degree of debug
- Example devicetrees for specific platforms (configurations)
- Kuiper-Linux
 - Preconfigured with all drivers installed
 - Supports
 - Raspberry Pi
 - Arduino form factor ARM based FPGA platforms such as
 - Intel/TerASIC DE10-Nano
 - Xilinx/Digilent Cora Z7
 - Most popular FMC FPGA carriers from Xilinx and Intel with ARM/ARM64 support
 - Zynq7000 (Zed board, ZC706, ZC702)
 - MPSoC (ZCU102)
 - SoC FPGA (A10Soc, A5Soc)

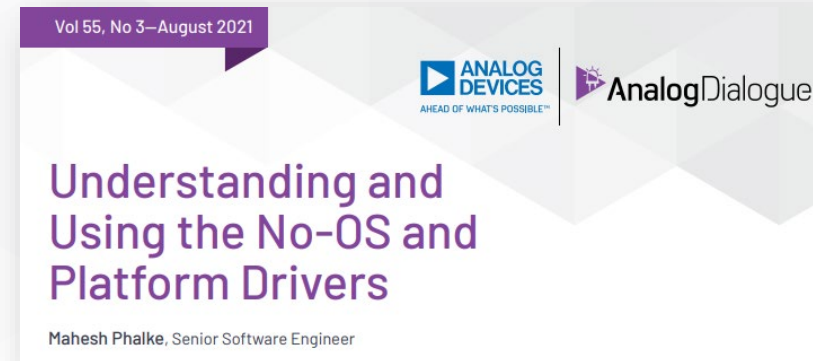


ADI-KUIPER-LINUX

LINUX DEVICE DRIVERS FOR ADI PERIPHERALS

► No-OS

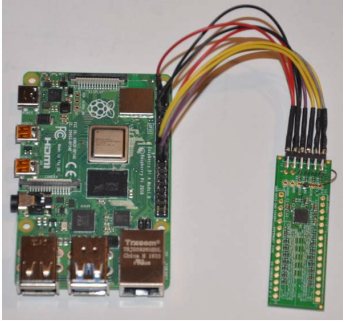
- Standalone drivers
- Targeted at RTOS style optimized systems
- Minimal requirements
- Example projects for specific platforms (configurations)
- <https://github.com/analogdevicesinc/no-OS>



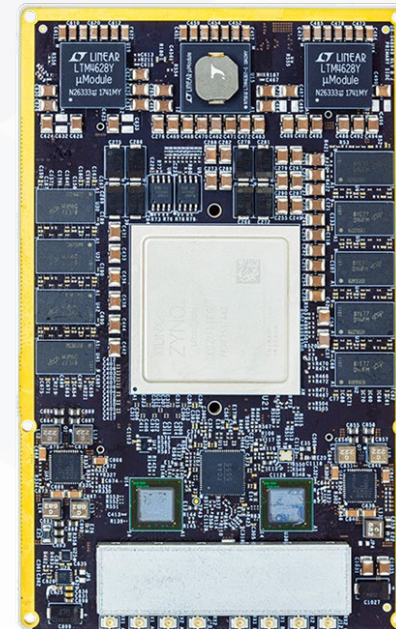
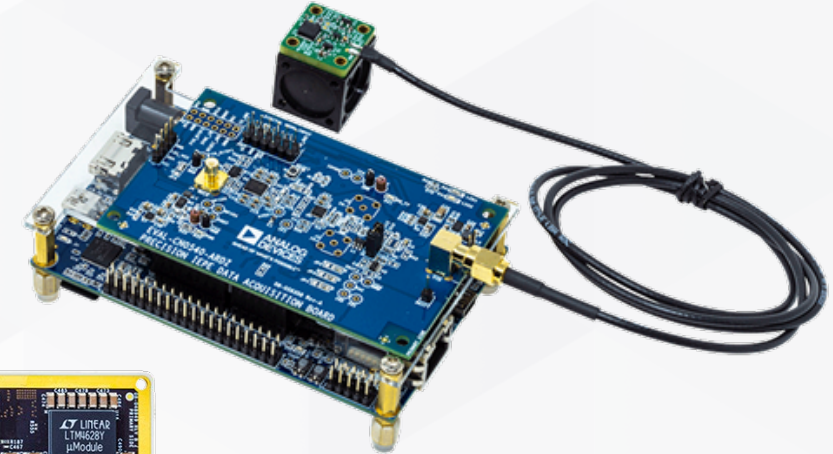
<https://www.analog.com/media/en/analog-dialogue/volume-55/number-3/understanding-and-using-the-no-os-and-platform-drivers.pdf>

IIO Runs on Various Hardware

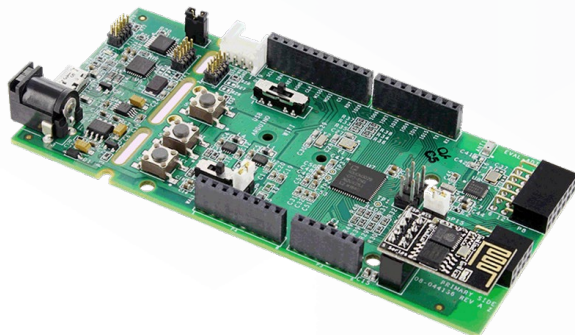
RPI Based



FPGA Based



Bare-metal Based



Python Wrapper for IIO:

► PyADI-IIO:

- <https://analogdevicesinc.github.io/pyadi-iio/>
- **PyADI-IIO is a python abstraction module for ADI hardware with IIO drivers to make them easier to use**
- “glue layer” between IIO and doing something useful in Python
- Pre-installed on ADI Kuiper Linux (ADI’s custom Raspberry Pi OS, with device drivers and utilities)



```
import adi

# Create radio
my_sdr = adi.Pluto()

# Configure properties
my_sdr.rx_lo = 2200000000
my_sdr.tx_lo = 2200000000

# Collect data
data = my_sdr.rx()
```

PYADI-IIO Examples:

Step 1: Import the “adi” library – all Analog Devices Python objects are here



```
import adi
```

Step 1: Import the "adi" library – all Analog Devices Python objects are here



```
import adi
```

```
my_ad5592r = adi.ad5592r(uri="ip:192.168.86.123") # Connect to AD5592r (on remote Pi)
```

Step 2: Create an Object, with the context you are using:

"local:" (read-only for standard users on RPi)

"ip:localhost" (most convenient when run on RPi)

"ip:192.168.86.123" (Your Pi's wifi/wired address)

"usb:1.2.3" (USB Context ID)

"serial:COM1" (Serial port number)

Step 1: Import the "adi" library – all Analog Devices Python objects are here



```
import adi
```


```
my_ad5592r = adi.ad5592r(uri="ip:192.168.86.123") # Connect to AD5592r (on remote Pi)
```

```
my_ad5592r.channel[1].raw = 1024 # Set to 1/4 scale
```

```
ch1_readback = my_ad5592r.channel[1] # Read back actual voltage
```

```
ch1_scale = my_ad5592r.channel[1].scale # Read channel scale (mV per Lsb)
```

```
print("Voltage: ", ch1_readback * ch1_scale)
```



Step 2: Create an Object, with the context you are using:

"local:" (read-only for standard users on RPi)

"ip:localhost" (most convenient when run on RPi)

"ip:192.168.86.123" (Your Pi's wifi/wired address)

"usb:1.2.3" (USB Context ID)

"serial:COM1" (Serial port number)

Step 3: Set attributes and transmit/receive data

PYADI-II0 Example: High Speed Precision DAC

AD3552R: Maximum Speed/Precision Mode for Maximum Accuracy

The AD3552R is a low drift, ultrafast, 16-bit accuracy, current output digital-to-analog converter (DAC) with adjustable voltage span ranges, and operates with a fixed 2.5 V reference. Multiple error checks have been included to identify V_{REF} failures and memory map corruption.



```
my_ad3552r_0 = adi.ad3552r(uri="ip:analog.local",  
                             device_name="axi-ad3552r-0")  
my_ad3552r_0.tx_enabled_channels = [0]  
fs = int(my_ad3552r_0.sample_rate)  
# Sine generation  
samples = customer_waveform_generator()  
# ... #  
my_ad3552r_0.tx_cyclic_buffer = True  
my_ad3552r_0.tx(samples)  
input("Press Enter to stop the stream...")
```

PYADI-IIO Example: High Speed ADC

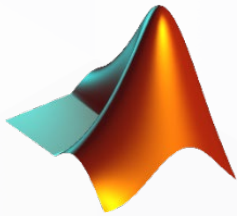
LTC2387-18: High Speed Data Acquisition

LTC2387-18 is ideal for high speed imaging and instrumentation application. No-latency operation is ideal for high speed control loops. Low distortion at high input frequencies enables scientific applications with wide dynamic range and signal bandwidth.



```
my_ltc2387 = adi.ltc2387(uri="ip:analog.local")
my_ltc2387.rx_buffer_size = 4096
my_ltc2387.sampling_frequency = 10000000
data = my_ltc2387.rx()
result = customer_analysis_function(data)
```

Or Run IIO in Matlab, with Very Similar Syntax



MATLAB

```
>> rx = sdr_rx('AD936x');
>> rx = sdr_rx('Pluto');
```

rx =

[comm.SDRRxPluto](#) with properties:

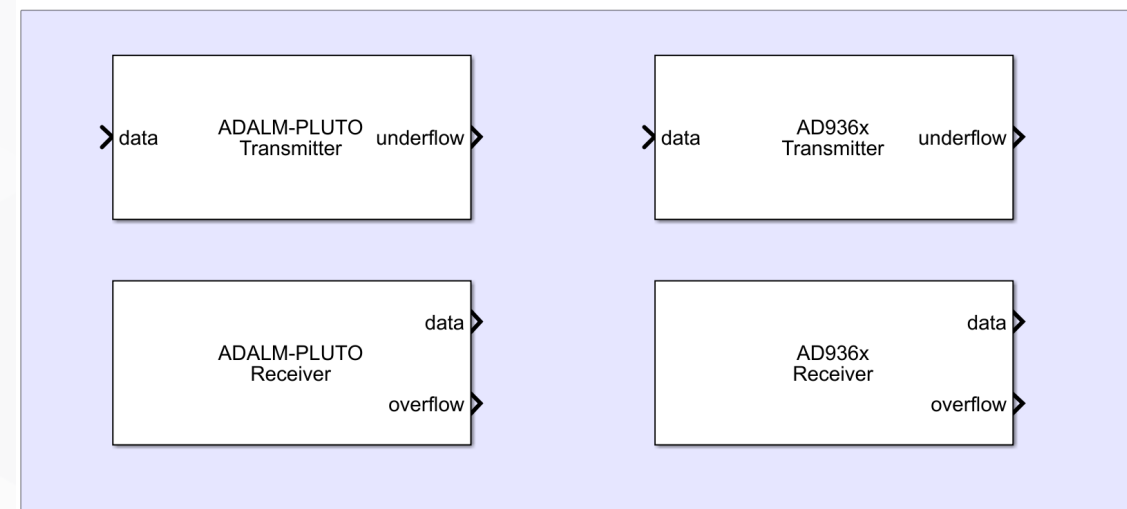
Main

```
    DeviceName: 'Pluto'
    RadioID: 'usb:0'
    CenterFrequency: 2.4000e+09
    GainSource: 'AGC Slow Attack'
    ChannelMapping: 1
    BasebandSampleRate: 1000000
    OutputDataType: 'int16'
    SamplesPerFrame: 20000
    EnableBurstMode: false
    ShowAdvancedProperties: false
```

Show [all properties](#)

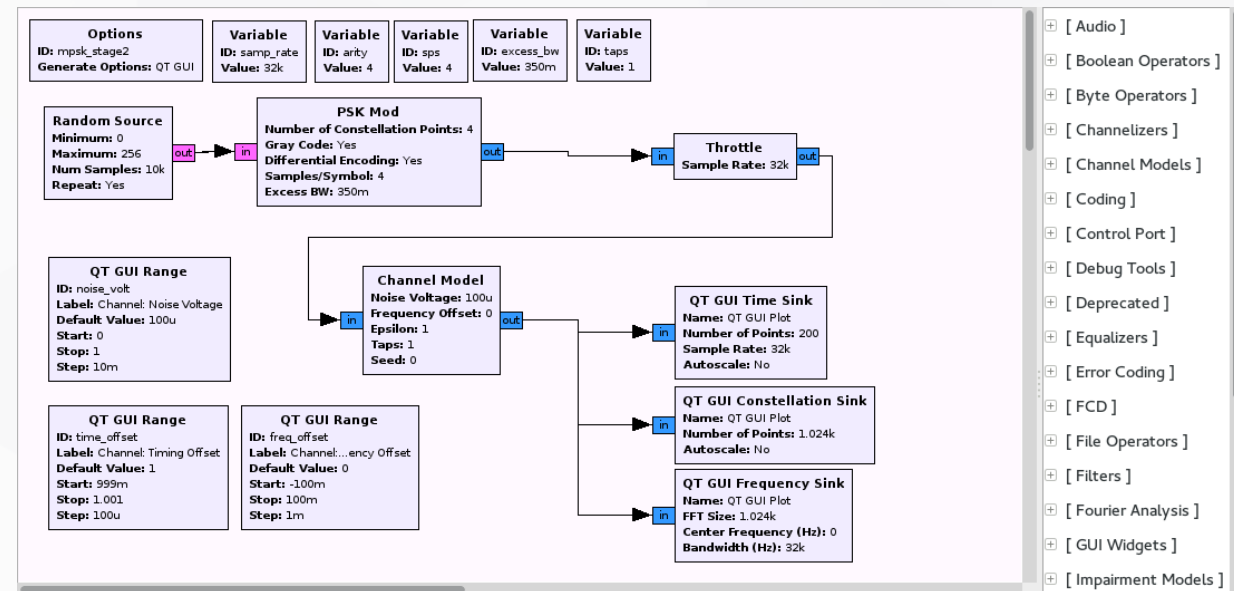


Simulink

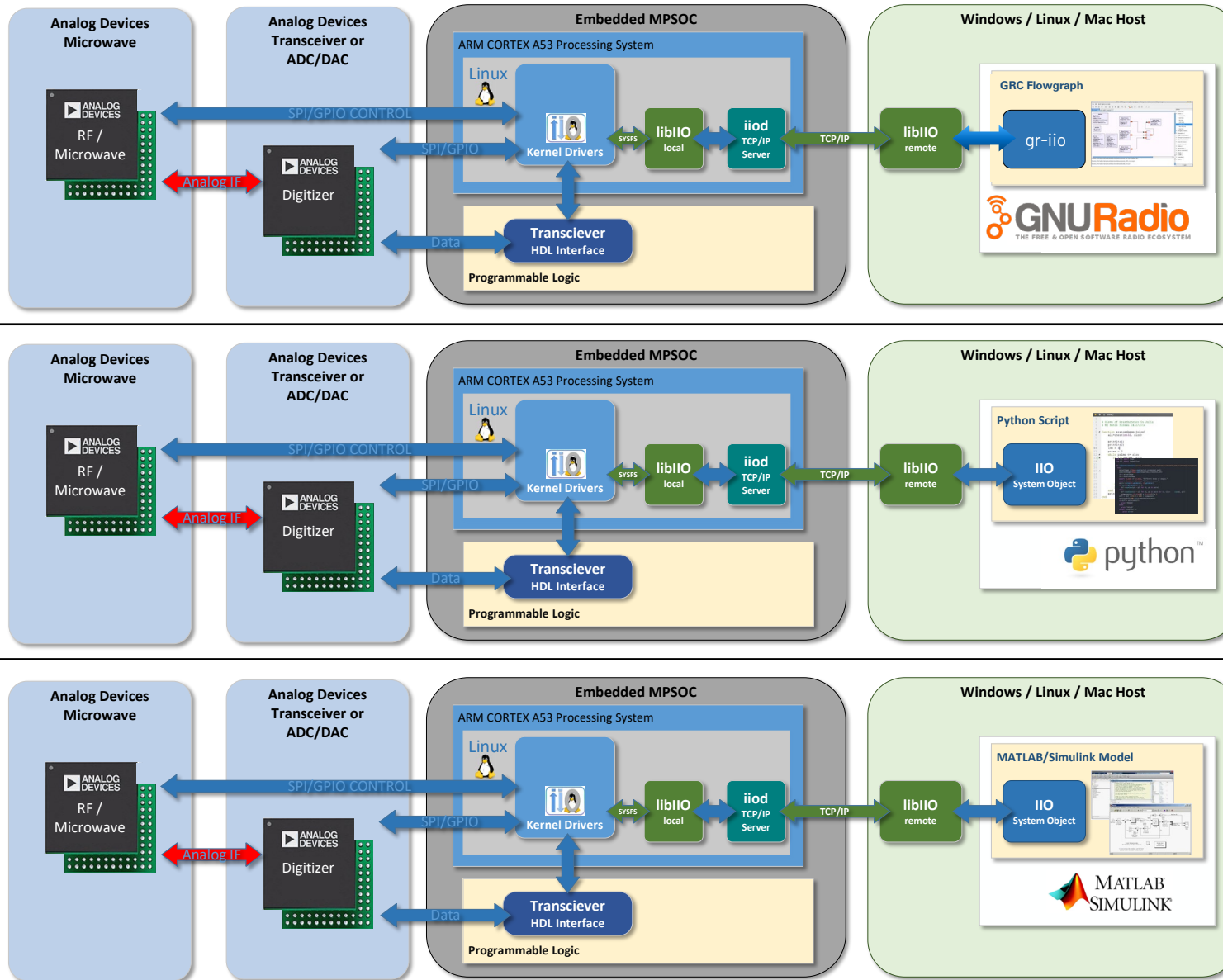


Or Run IIO in GNU Radio

- ▶ Signal-processing development environment
- ▶ Opensource software
- ▶ Focused on Software-Defined-Radio
- ▶ Many standard processing blocks available
- ▶ Processing pipelines are assembled as flowgraphs
- ▶ IIO Source and Sink blocks
- ▶ Can either use local or network backend
- ▶ Can also add PYADI-IIO with Python Block

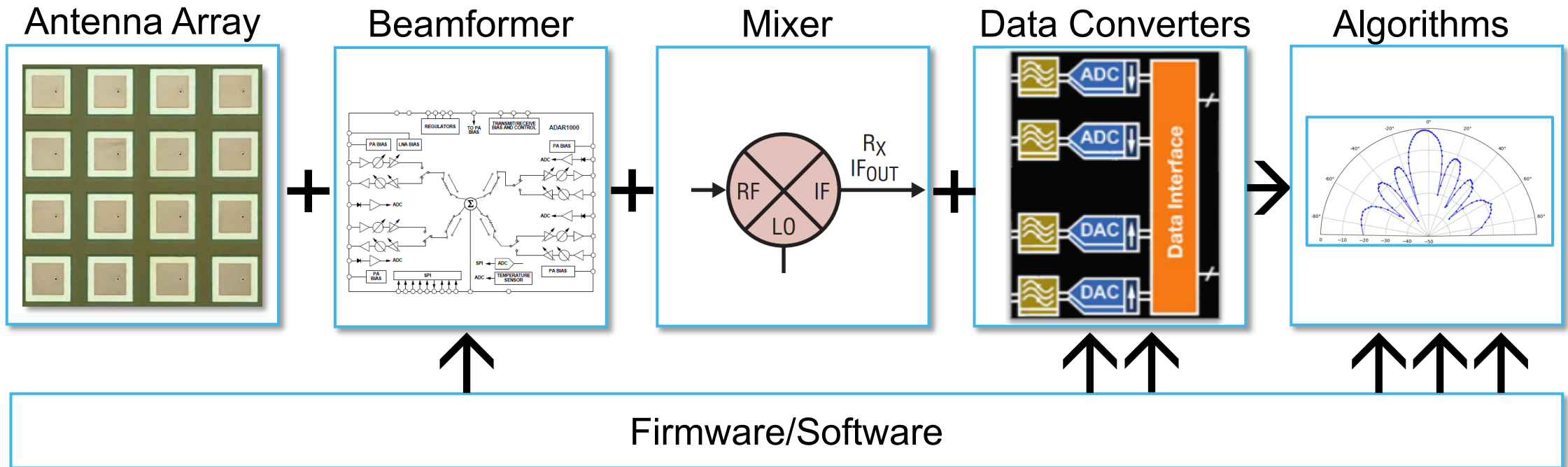


Single cohesive software solution

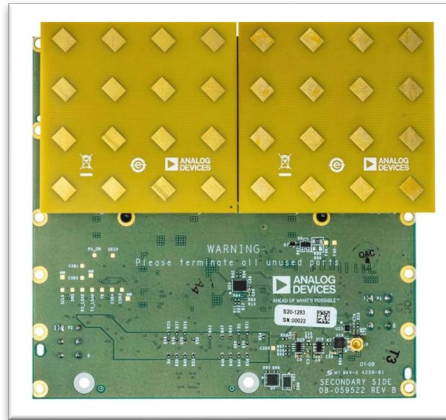


How do We Apply this to Phased Array Prototyping?

Change the Hardware → Keep the Software the Same!



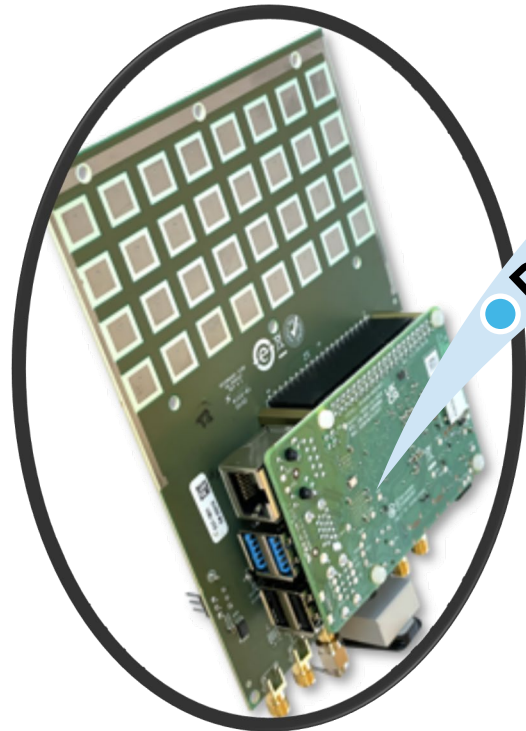
From Concept → Prototype → Production



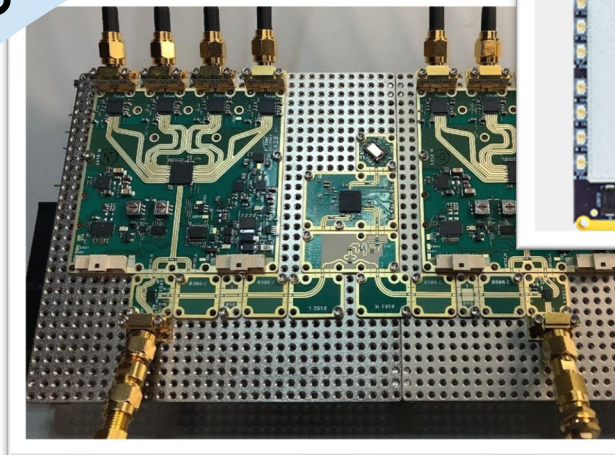
**X-Band Phased
Array Platform**



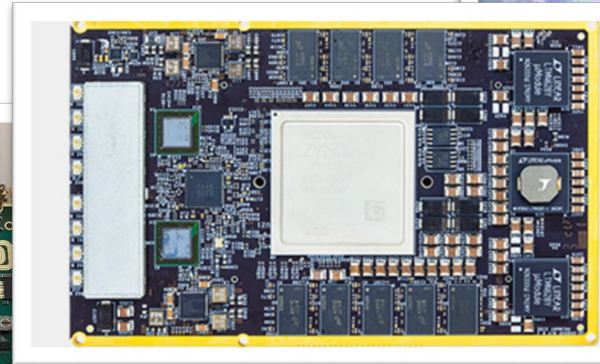
Final Production



Phaser



X-Microwave Prototyping System



Production Ready SOMs

Production

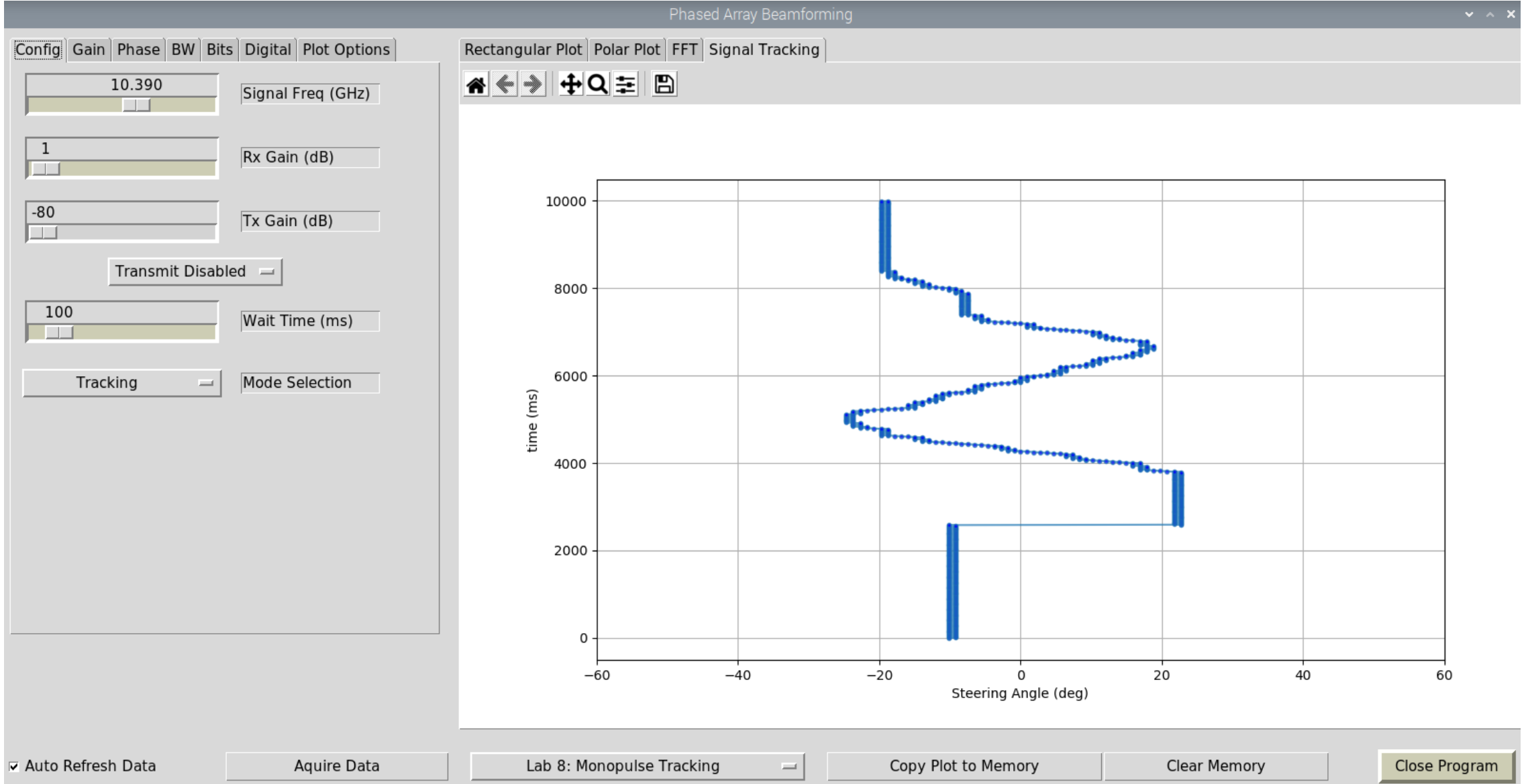
Prototype

**Same software
can be used at
each stage!**

Live Demo

What is the Algorithm We are Developing?

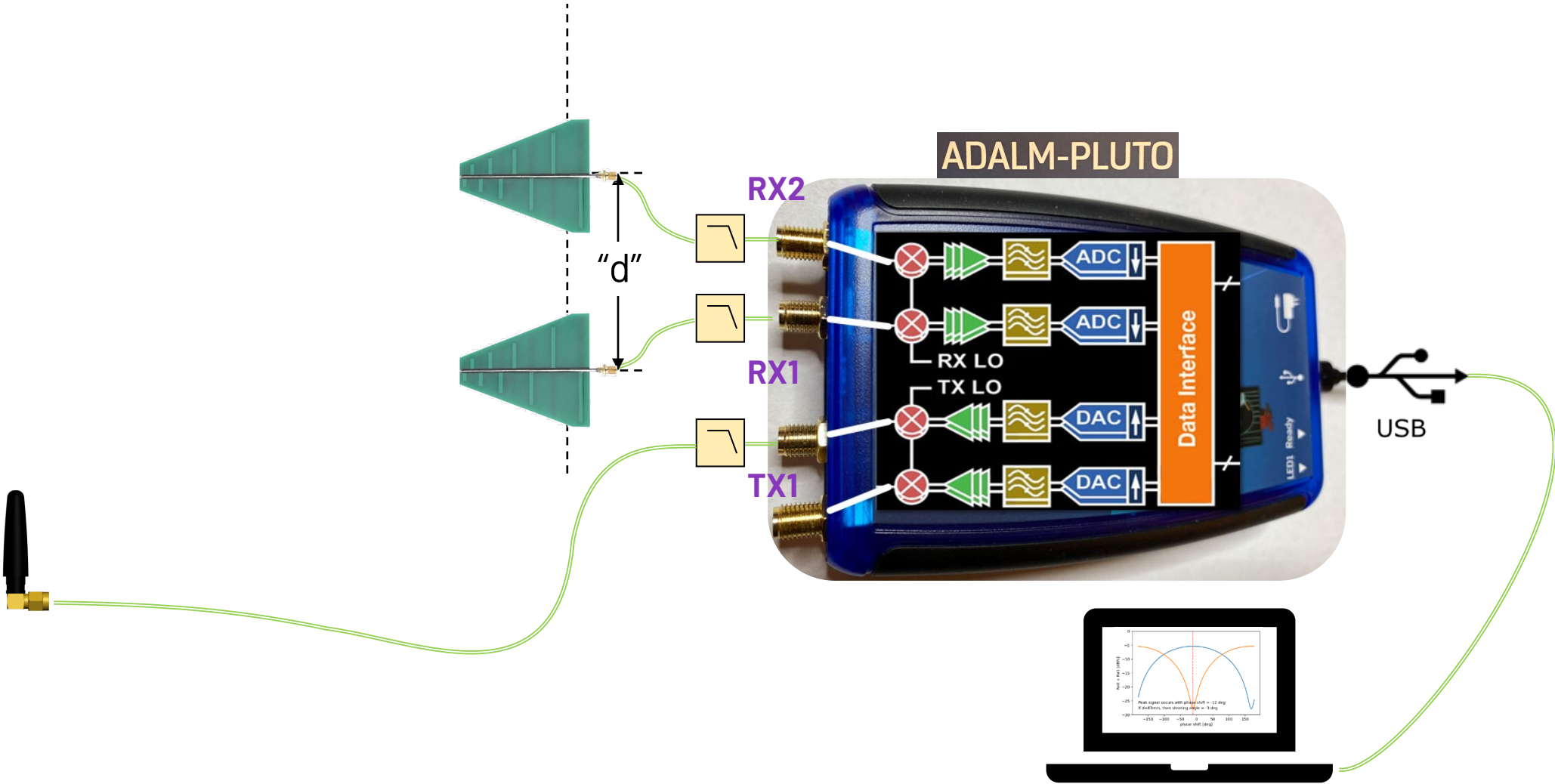
Monopulse Tracking Live Demo



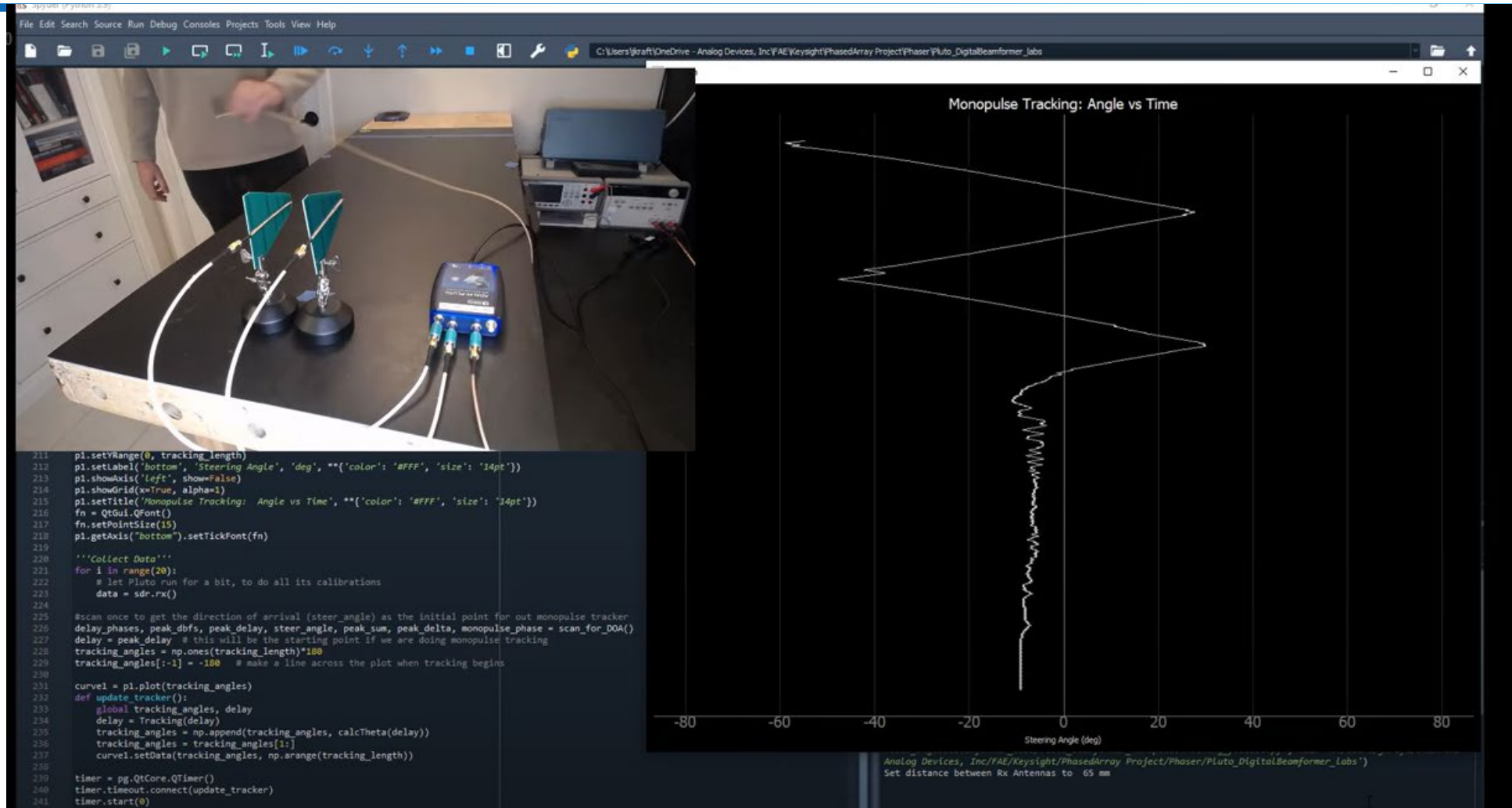
First: CRAWL

Get Started Building Something

Pluto Digital Beamformer Setup



Monopulse Tracking with Pluto

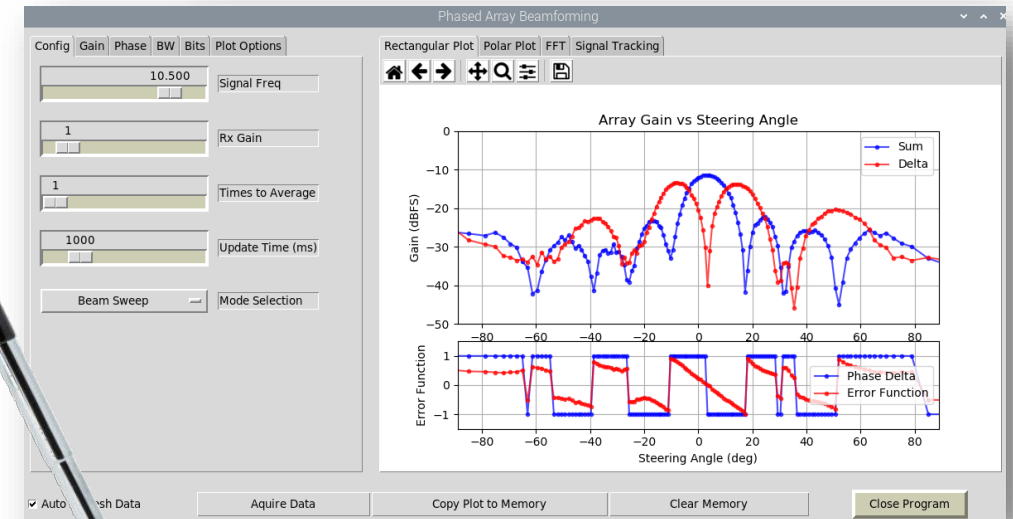
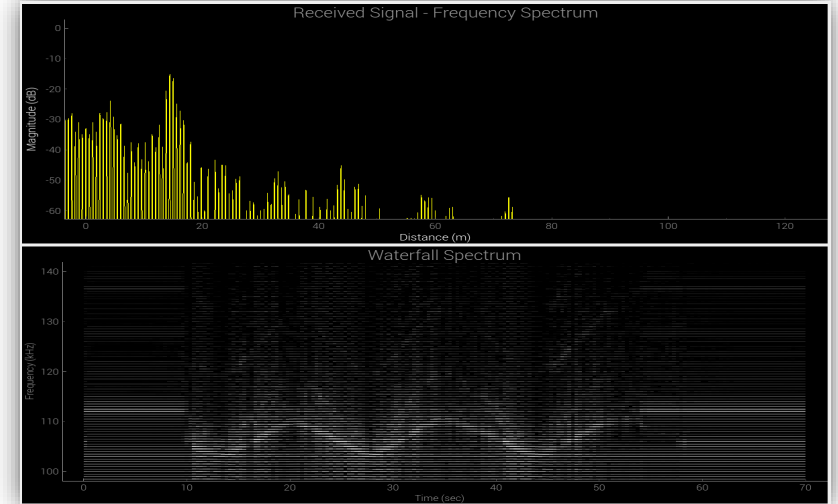


<https://youtu.be/XP80WMDHfOQ>

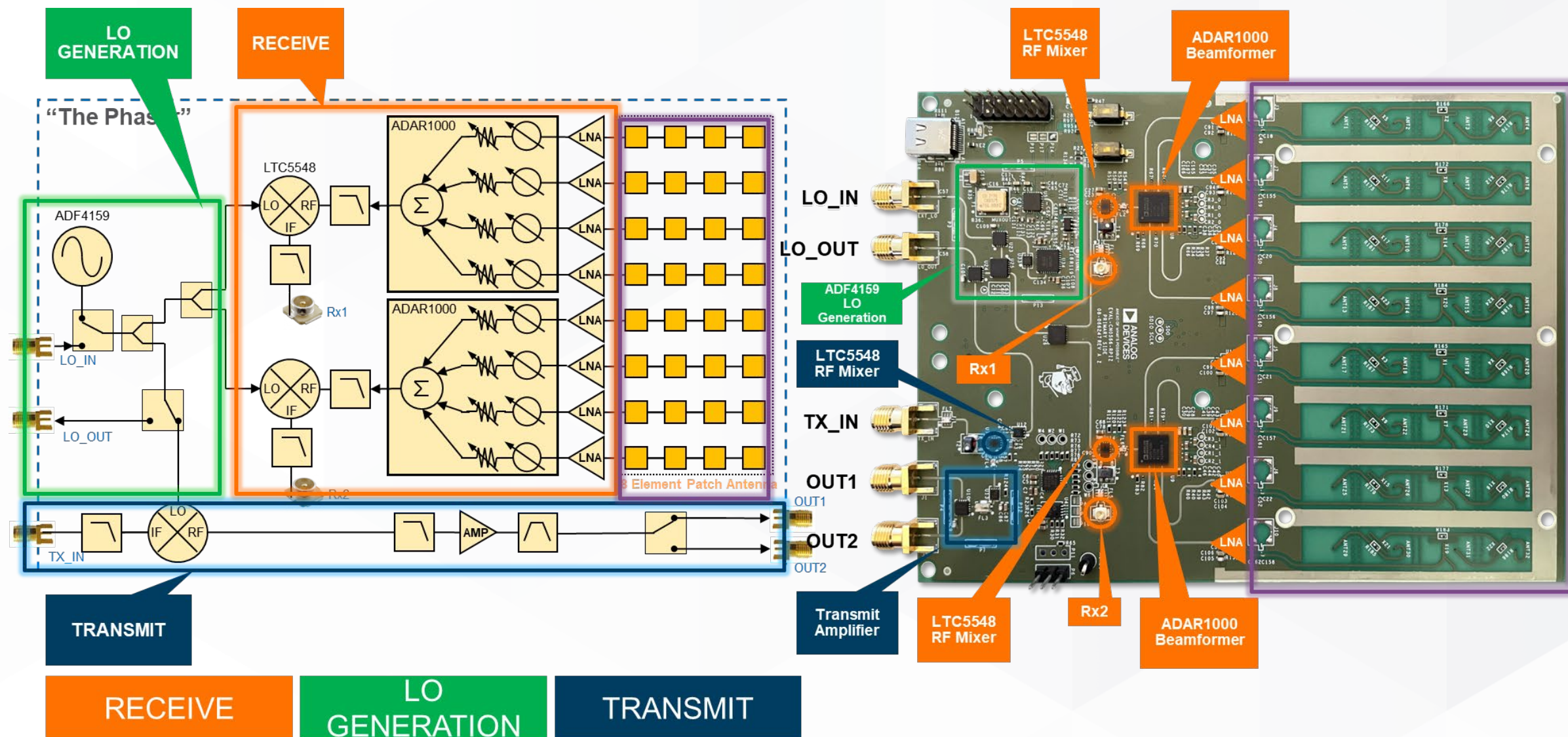
The Phaser: *X Band Phased Array RADAR System*



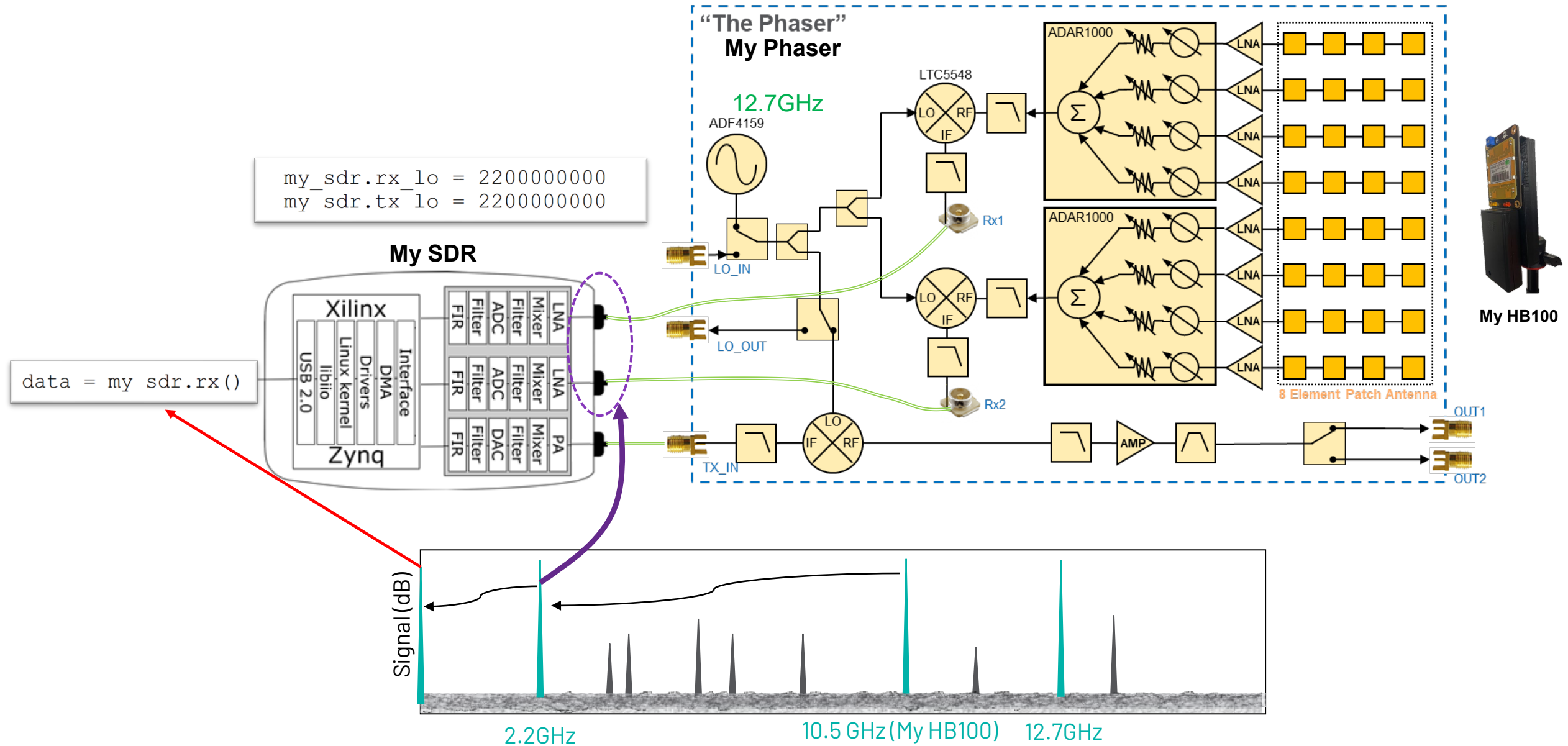
- ▶ **Phased Array Education/Prototyping**
- ▶ Comms: 10-11 GHz operation
- ▶ Radar: 500MHz BW FMCW Chirps
- ▶ 8 channel Receive, 2 channel Transmit
- ▶ Open source software, hardware
- ▶ Price: ~\$2500 for the entire kit
- ▶ www.analog.com/cn0566
- ▶ wiki.analog.com/phaser



Phaser Block Diagram



What can we do with Python and Pyadi-iiio?



What does the Python look like to Control Phaser?

```
import adi

# Create Phaser object
my_phaser = adi.cn0566(uri="ip:phaser.local", rx_dev=my_sdr)

# Set all ADAR1000 channels to phase = 0 deg
# and apply a Blackman taper to the array
gain_list = [8, 34, 84, 127, 127, 84, 34, 8] #Blackman taper
for i in range(0, 8):
    my_phaser.set_chan_phase(i, 0)
    my_phaser.set_chan_gain(i, gain_list[i])
```

Raspberry Pi's IP address → `ip:phaser.local`

Pluto object we created earlier → `my_sdr`

► Full example script here:

https://github.com/analogdevicesinc/pyadi-iio/blob/cn0566_dev/examples/cn0566/cn0566_minimal_example.py

Phaser Snippets

Set up ADAR1000
element mapping

```
# Connect to Pluto SDR
my_sdr = adi.ad9361(uri="ip:192.168.2.1")

# Connect to devices on phaser board
my_adar1000_array = adi.adar1000_array(
    uri="ip:phaser.local",
    chip_ids=["BEAM0", "BEAM1"],
    device_map=[[1], [2]],
    element_map=[[1, 2, 3, 4, 5, 6, 7, 8]],
    device_element_map={
        1: [7, 8, 5, 6], # i.e. channel2 of device1 (BEAM1),
        2: [3, 4, 1, 2], # maps to element 8
    },)
my_adf4159 = adi.adf4159(uri="ip:phaser.local")
my_gpios = adi.one_bit_adc_dac(uri="ip:phaser.local")
```

Pluto's IP address

Raspberry Pi's IP address

Receive data, sum
channel 0 and 1, take
FFT, then plot

```
my_sdr.rx_lo = int(2.2e9) # Downconvert by 2GHz # Receive Freq
my_sdr.filter = "LTE20_MHz.ftr" # Handy filter for wideband measurements
my_adf4159.frequency = (
    int(10.525e9) + # HB100 frequency
    my_sdr.rx_lo - # SDR's LO (2.2 GHz)
    int(1e6)) // 4 # add 1 MHz offset to avoid DC
                  # we're using VCO's /4 output

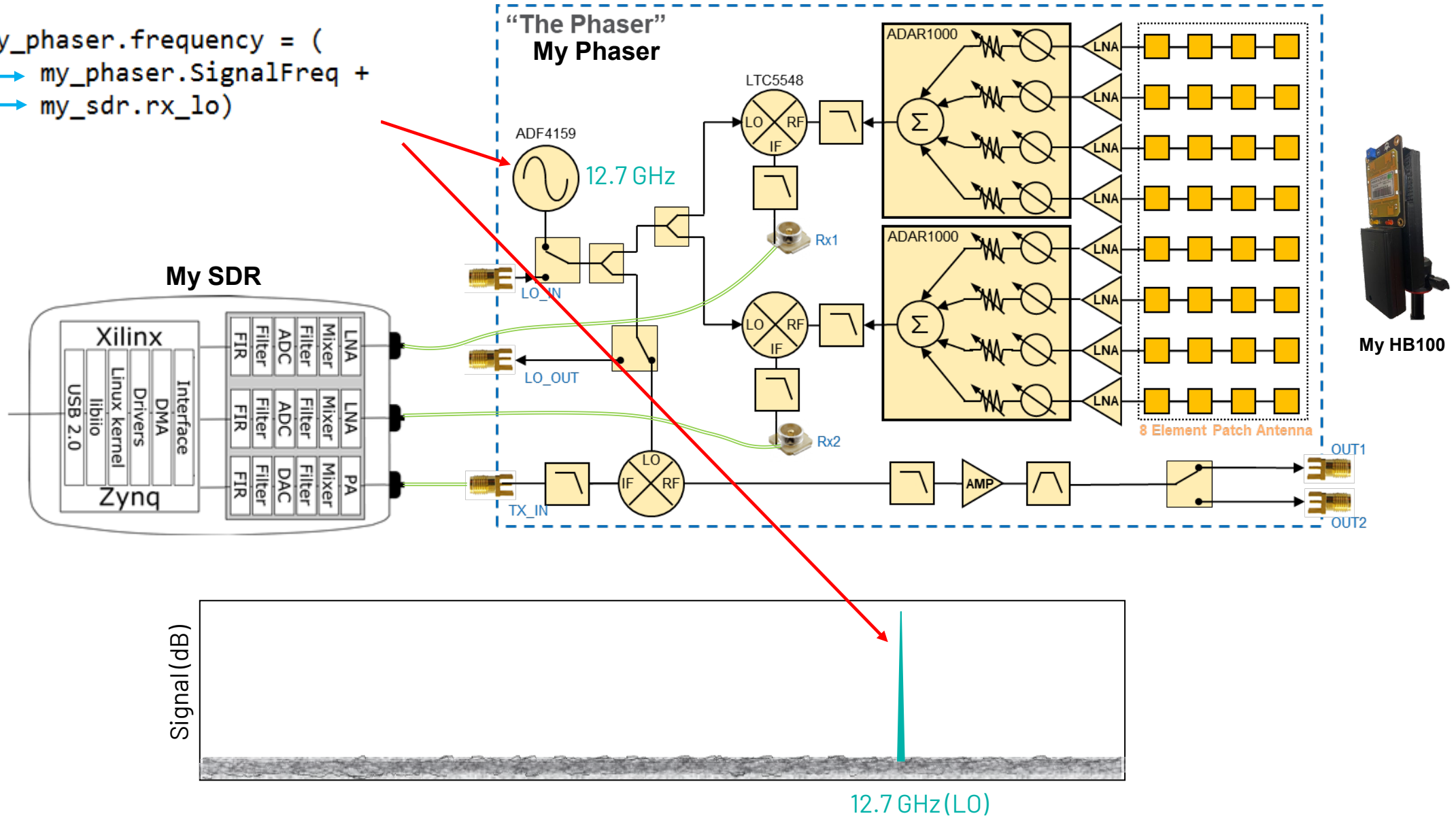
# Capture data!
data = my_sdr.rx()
# Add I and Q for calculating spectrum
data_sum = data[0] + data[1]

# spec_est is a simple estimation function that applies a
# window, takes the FFT, scales and converts to dB.
ampl, freqs = spec_est(data_sum, 30e6, ref=2 ^ 12, plot=False)

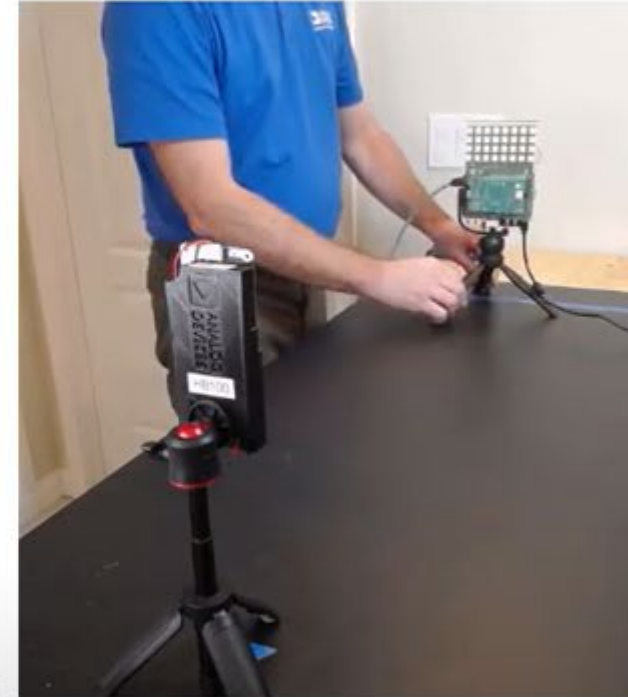
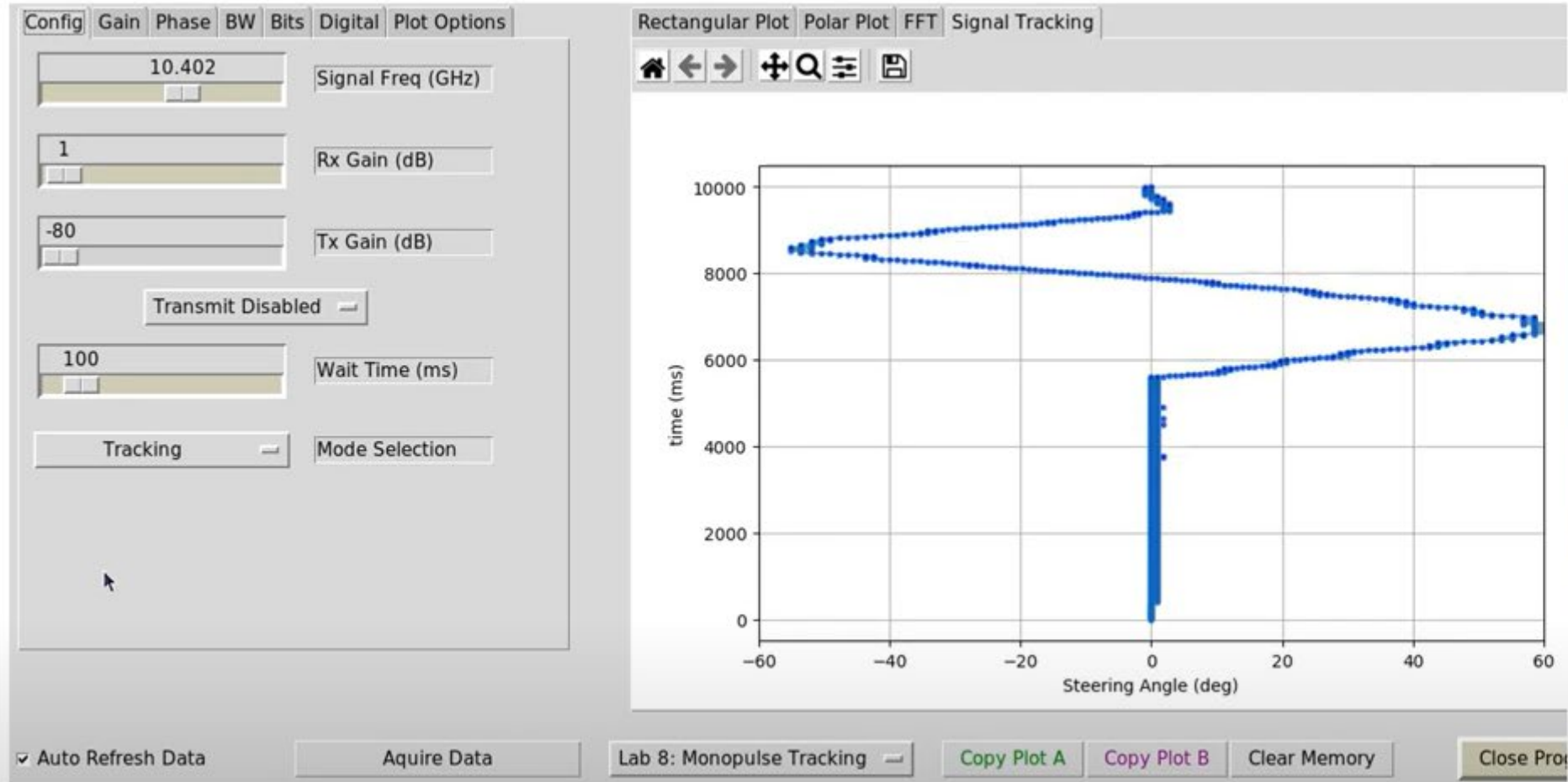
# Now... Plot!
# (do a web search on "matplotlib")
```

What can we do with Python and Pyadi-iiio?

```
my_phaser.frequency = (  
10.5 GHz → my_phaser.SignalFreq +  
2.2 GHz → my_sdr.rx_lo)
```



Monopulse Tracking with Phaser

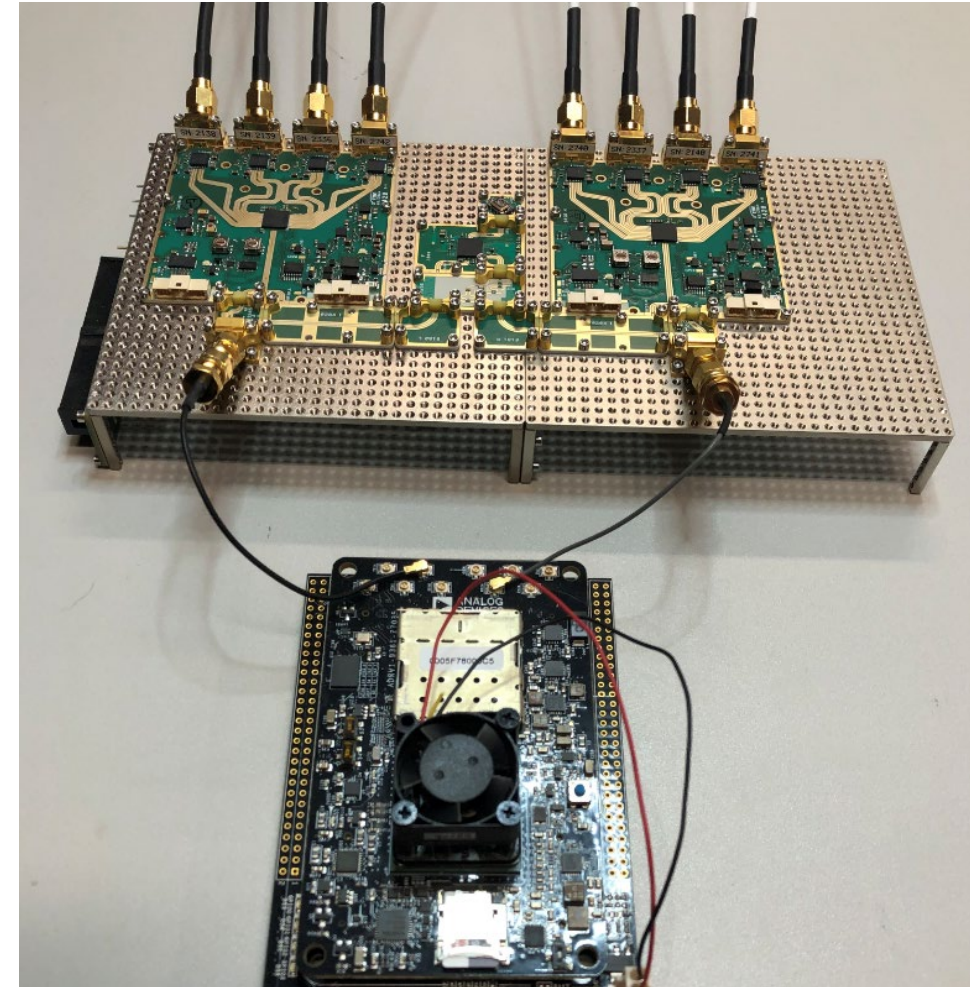
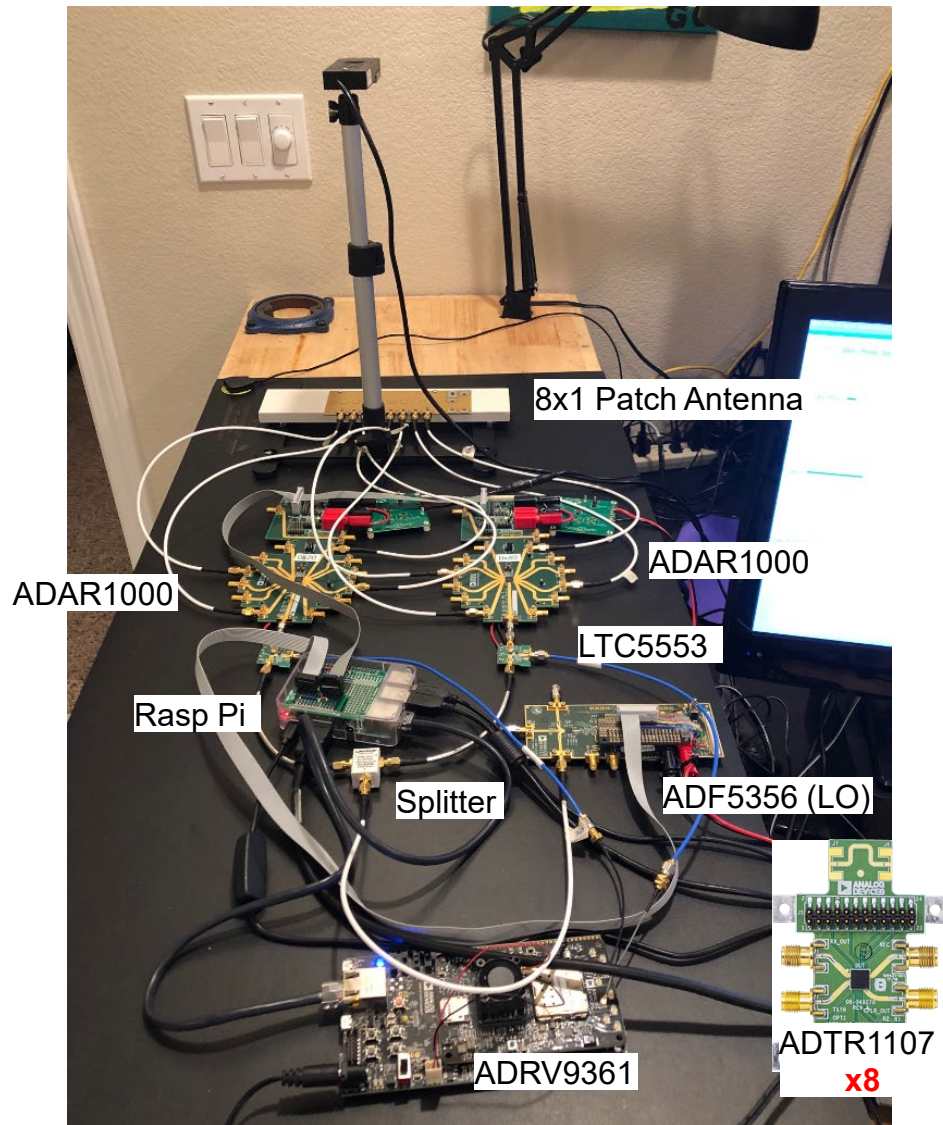


<https://youtu.be/hZDopcNjVzY>

Walk

*Add more elements, reconfigure the hardware,
but don't change the software!*

8 Element Setup Converted to X Microwave System



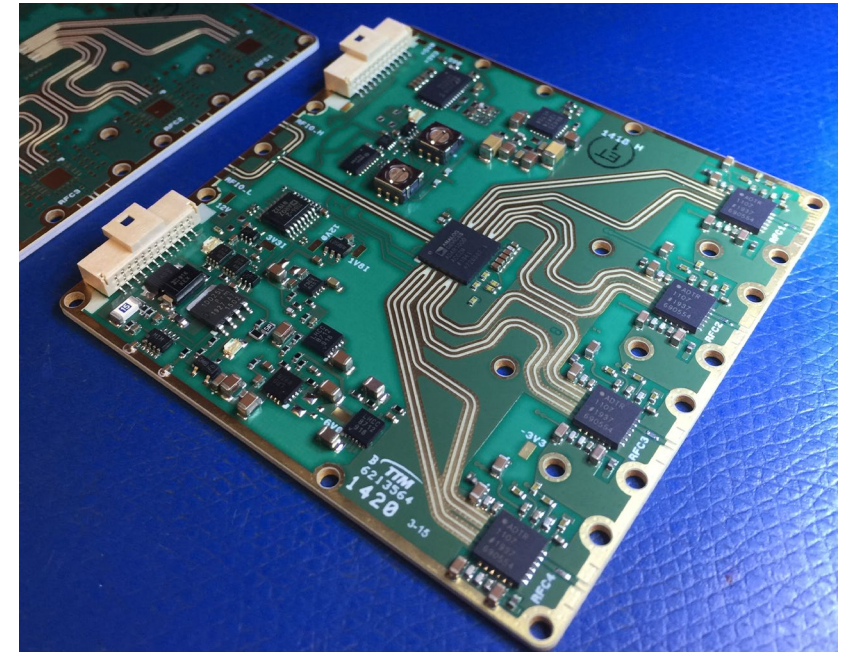
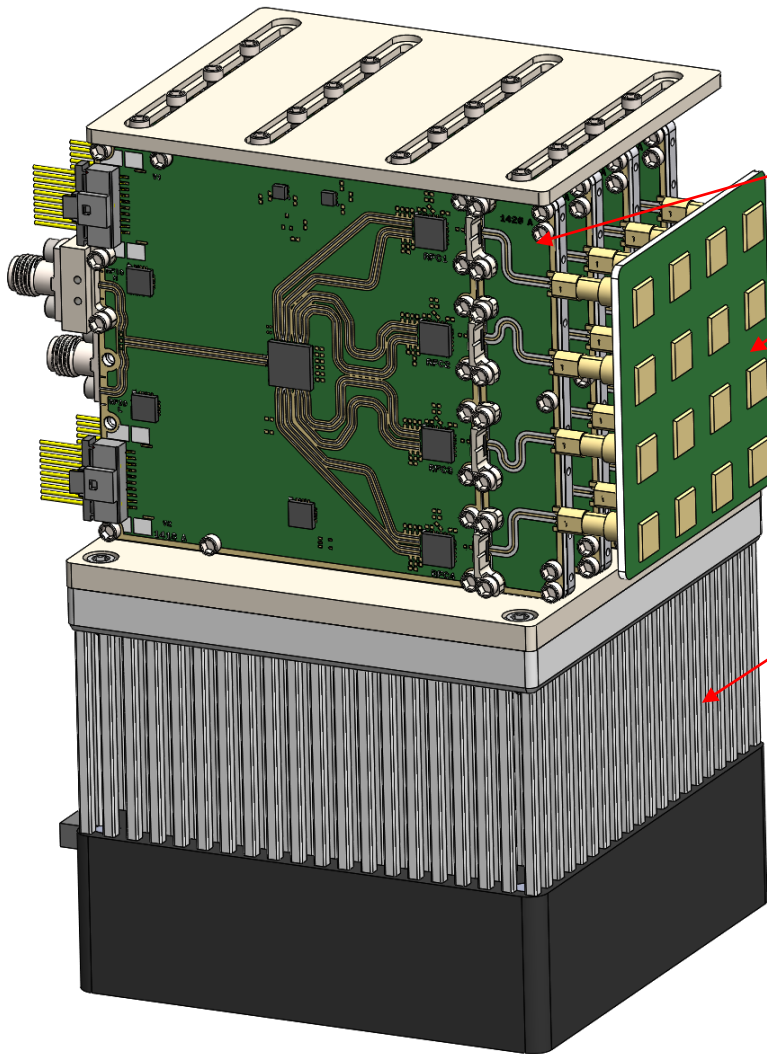
Stack ADAR1000 Modules Together for the Phased Array Cube:

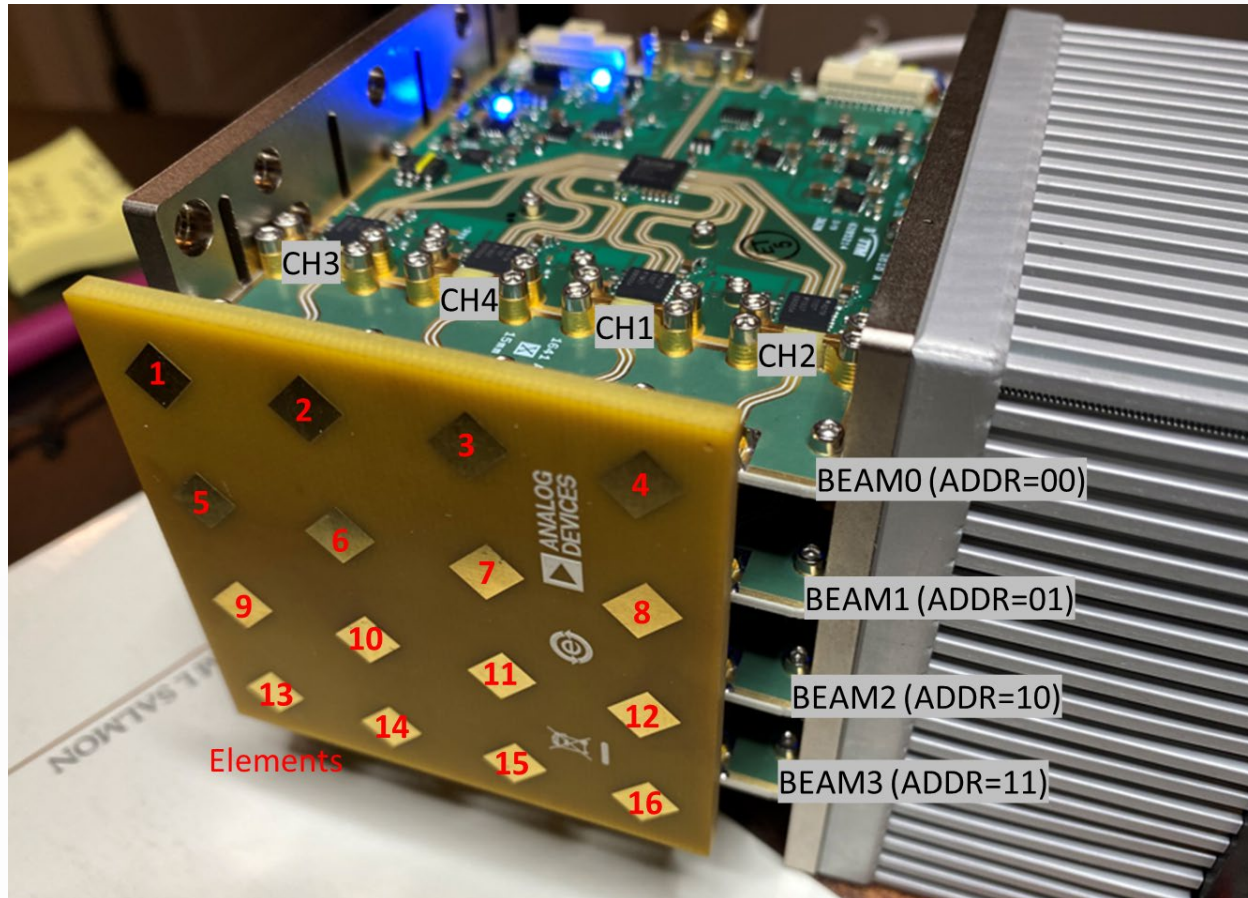
Stack 4 together to create a 4x4 array

Interposer board to fit whatever lattice spacing

Antenna snaps on

Heatsink





```
array = adi.adar1000_array(
    chip_ids=["BEAM0", "BEAM1", "BEAM2", "BEAM3"],
    device_map=[
        [1],
        [2],
        [3],
        [4]
    ],
    element_map=[
        [1, 2, 3, 4],
        [5, 6, 7, 8],
        [9, 10, 11, 12],
        [13, 14, 15, 16]
    ],
    device_element_map={
        1: [3, 4, 1, 2],
        2: [7, 8, 5, 6],
        3: [11, 12, 9, 10],
        4: [15, 16, 13, 14]
    }
)
```

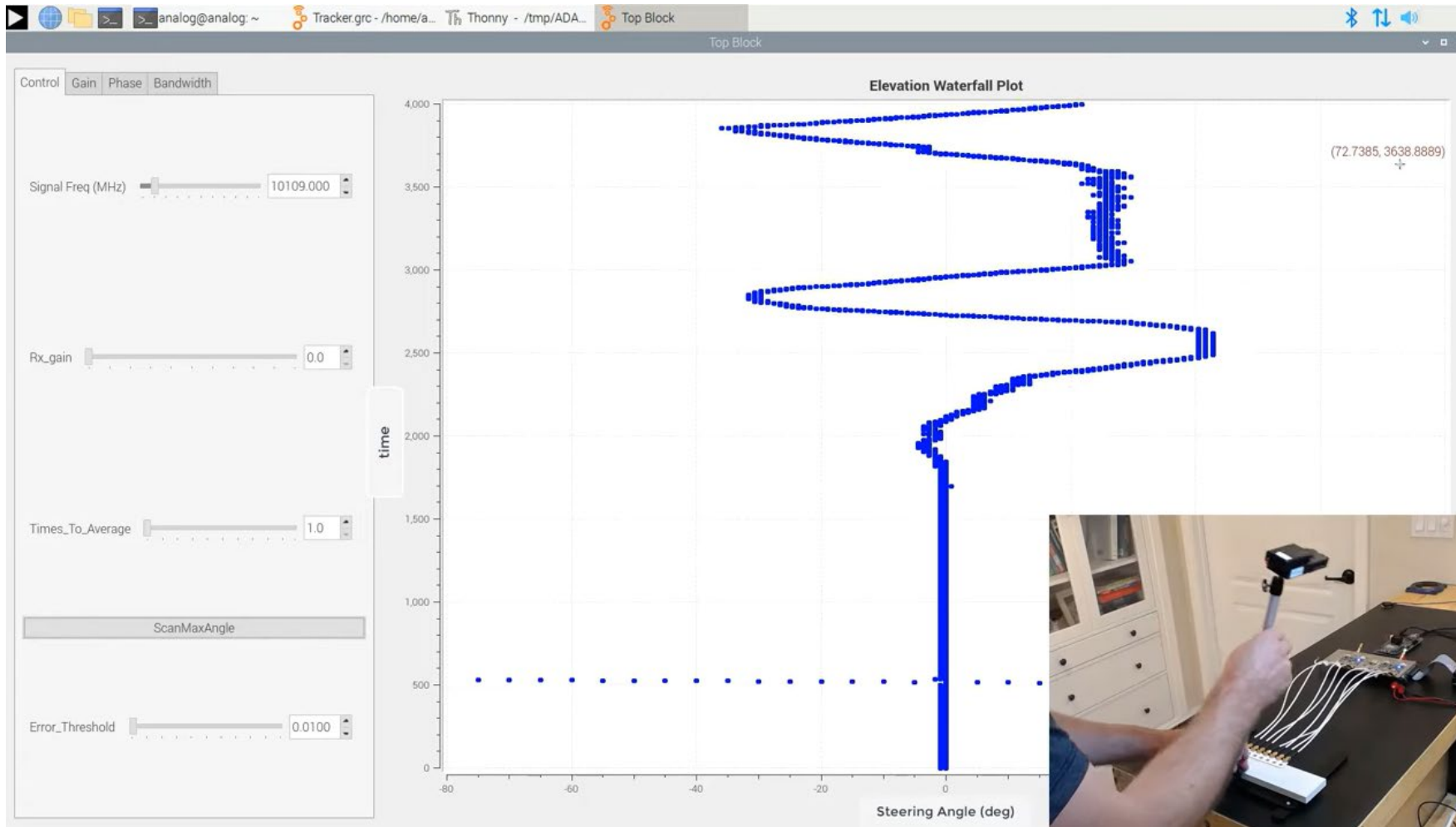
Now, each element can be addressed directly. **There is no need to keep track of which ADAR1000 each element belongs to!**

- For example:
 - el6 = array.elements.get(6)
 - el6.rx_phase, or el6.rx_gain, or el6.row, etc.

Beam Steering is very easy then:

```
for element in array.elements.values():
    column_phase = element.column * azimuth_phi
    row_phase = element.row * elevation_phi
    element.rx_phase = column_phase + row_phase
```

Monopulse Tracking with X Microwave



<https://youtu.be/0hnWfTvETcU>

Run

Use Larger Arrays Closer to Your Production
Version.
But with Minimal Changes to the Software!!

Stingray

X/Ku Band Phased Array RF Prototyping Board

► A 32-Channel Analog Phased Array RF Prototyping board consisting of 8xADAR1000 and 32xADTR1107

► RF

- RF IN, RF OUT (no frequency translation)
- Separate RFIO for Each Beamformer IC
- Lattice Spacing = 590mils/15mm (10 GHz)
- SMPM Connectors
- Stand-Alone RF Detector/ADC for Calibration

► Power Management

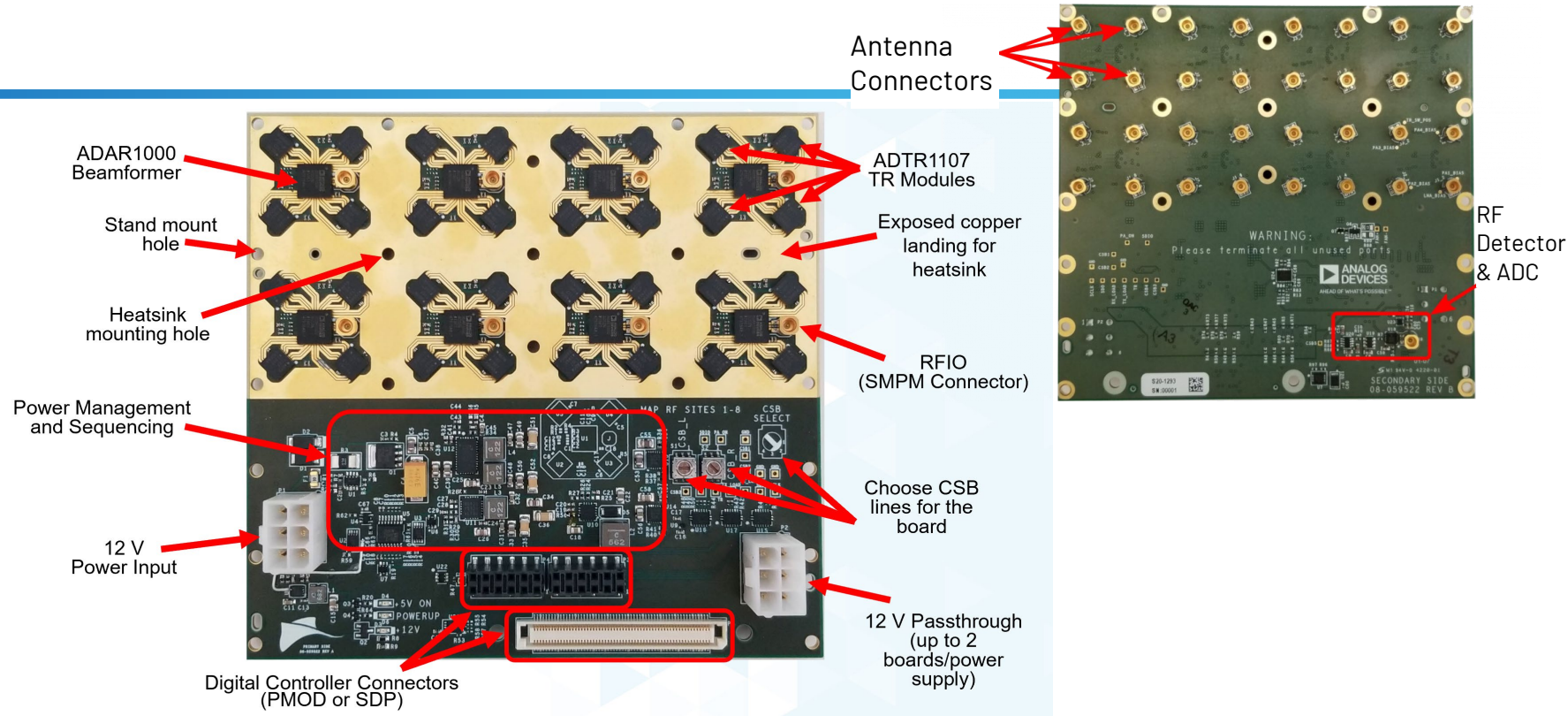
- 12V DC Power Input
- Integrated Power Management and Protective Sequencing

► Control

- PMOD and SDP Connectors for Programming

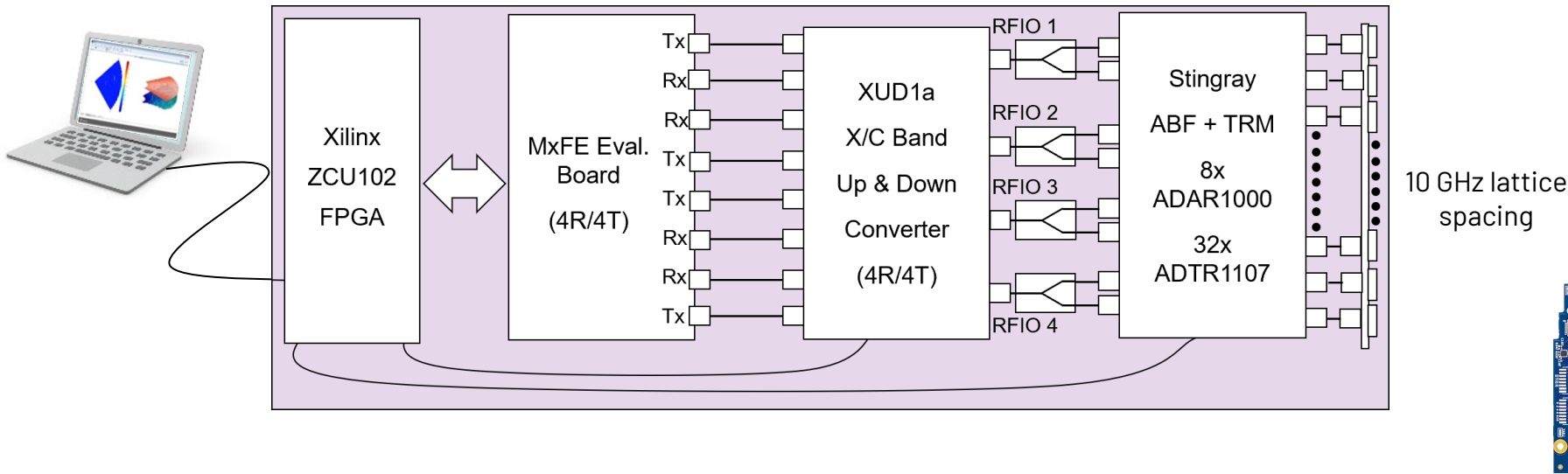
► Thermal Management

- Component-Side Heatsink with Cut-Outs for Access to RF I/O connectors

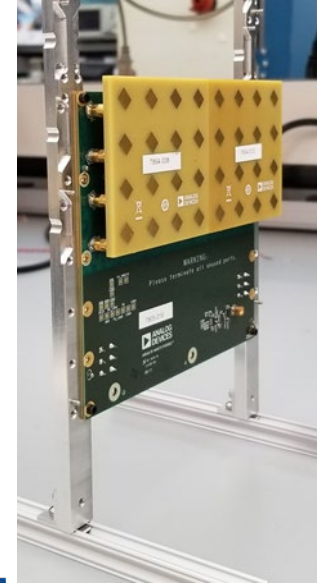


ADAR1000/ADTR1107 Combo

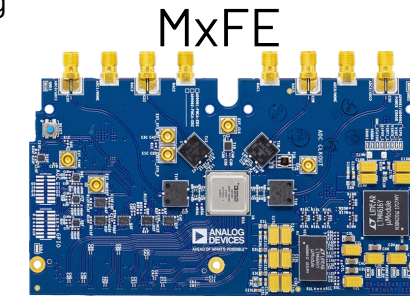
X Band (8 – 12 GHz) Phased Array Prototyping and Development System



XUD1a



Stingray



MxFE

Flexible X Band Prototyping/Development System

- Snap-On Antenna Board with 10 GHz Lattice Spacing
- Stingray 32-Channel Beamformer Board with T/R Modules and Lattice-Spaced Connectors
- XUD1a Up/Down Conversion Between X Band and C Band
- MxFE Evaluation Board Interfacing to the FPGA Board (ZCU102) and Matlab

- Applications
 - Phased Array Radar
- Full Signal Chain solution
 - Mix and Match Boards to Optimize
- Platform: Release FY3Q22
- Stingray: Limited Samples Now, release FY2Q22
- XUD1A: Limited Samples Now, release FY2Q22
- MxFE Eval: Released

Conclusion

Change the Hardware → Keep the Software the Same!

