Universal High-Bandwidth SDR Transceiver for RFSoC4x2 and GNU Radio

Marius Šiaučiulis, Louise H. Crockett and Robert W. Stewart

github.com/marsiau
marius.siauciulis@strath.ac.uk
github.com/strath-sdr/rfsoc_qsfp_offload

Contents

StrathSDR (and Neutral Wireless)

- **StrathSDR** = University of **Strath**clyde **S**oftware **D**efined **R**adio
- StrathSDR is composed of around 20 staff, including:
  - 4 senior academics and academic-related staff
  - 8 research staff
  - 10 PhD students
  - Interns and project students

(some of!) the StrathSDR team, Dec 25.

Our spin-out company, Neutral Wireless, also has additional staff.

Neutral Wireless is a technology company specialising in 5G private networks, broadcast connectivity, SDR, consulting and training.

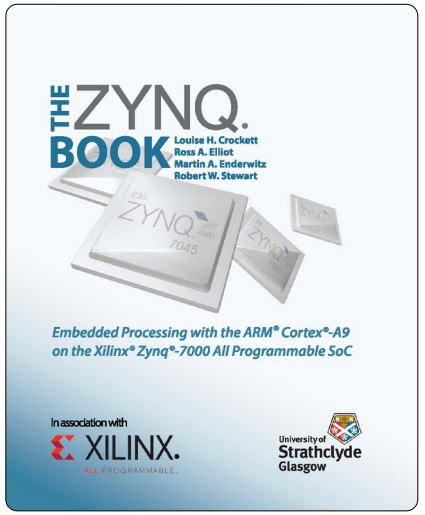
Book Development…

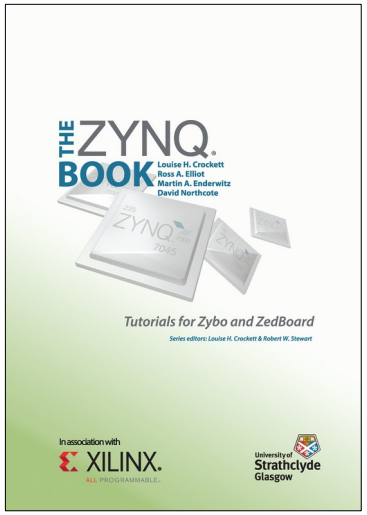
With **Xilinx** then **AMD**:

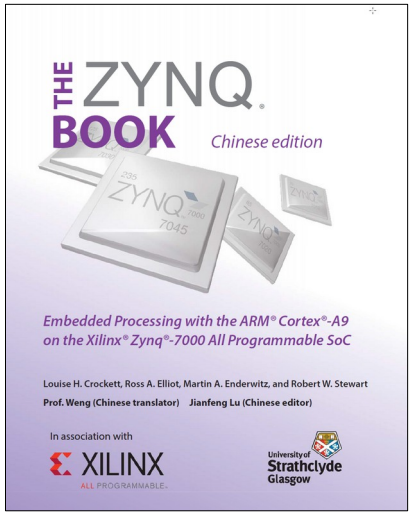
- The Zynq Book (2014)
  - The Zynq Book Tutorials
  - The Zynq Book Chinese Edition
- Exploring Zynq MPSoC (2019)
- Software Defined Radio with Zynq UltraScale+ RFSoC (2023)

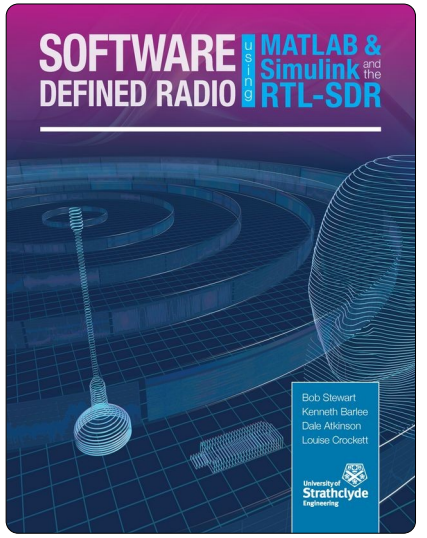
Also:

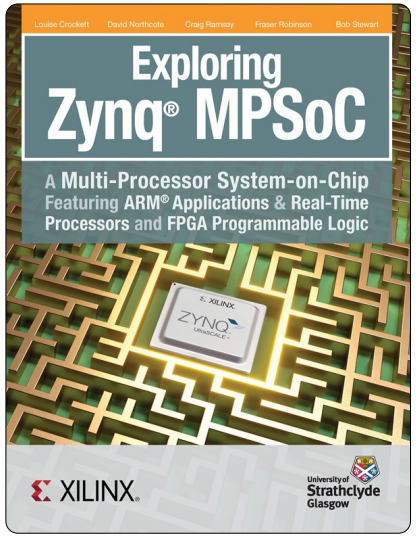
- Software Defined Radio with MATLAB, Simulink and the RTL-SDR (2015)









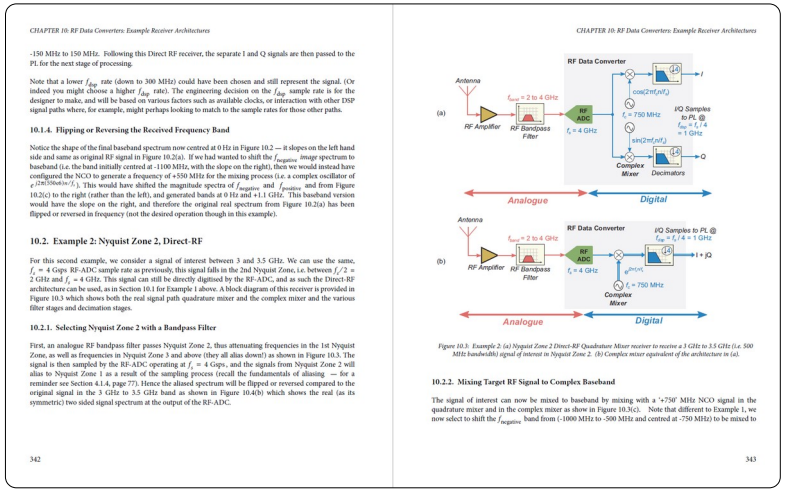


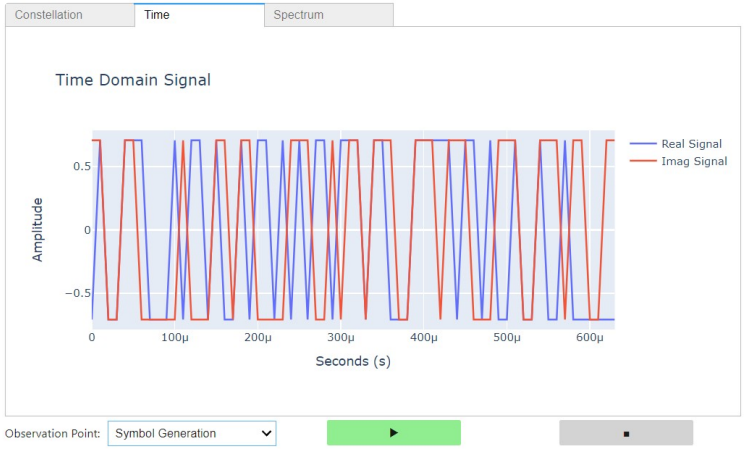
RFSoC Learning and Development Resources

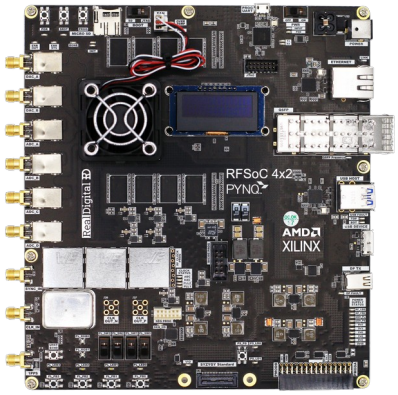
StrathSDR

- For anyone interested in Zynq RFSoC and SDR implementation, we have recently published a book on this topic with AMD.
- The book is available as a free PDF download as well as in hardback and softback from Amazon and other retailers.
- It is accompanied by a GitHub repository with Jupyter notebook tutorials, examples, and open-source reference designs for supported development boards, including:
  - Signal processing and comms principles
  - Radio transmitter and receiver designs
  - Frequency planning

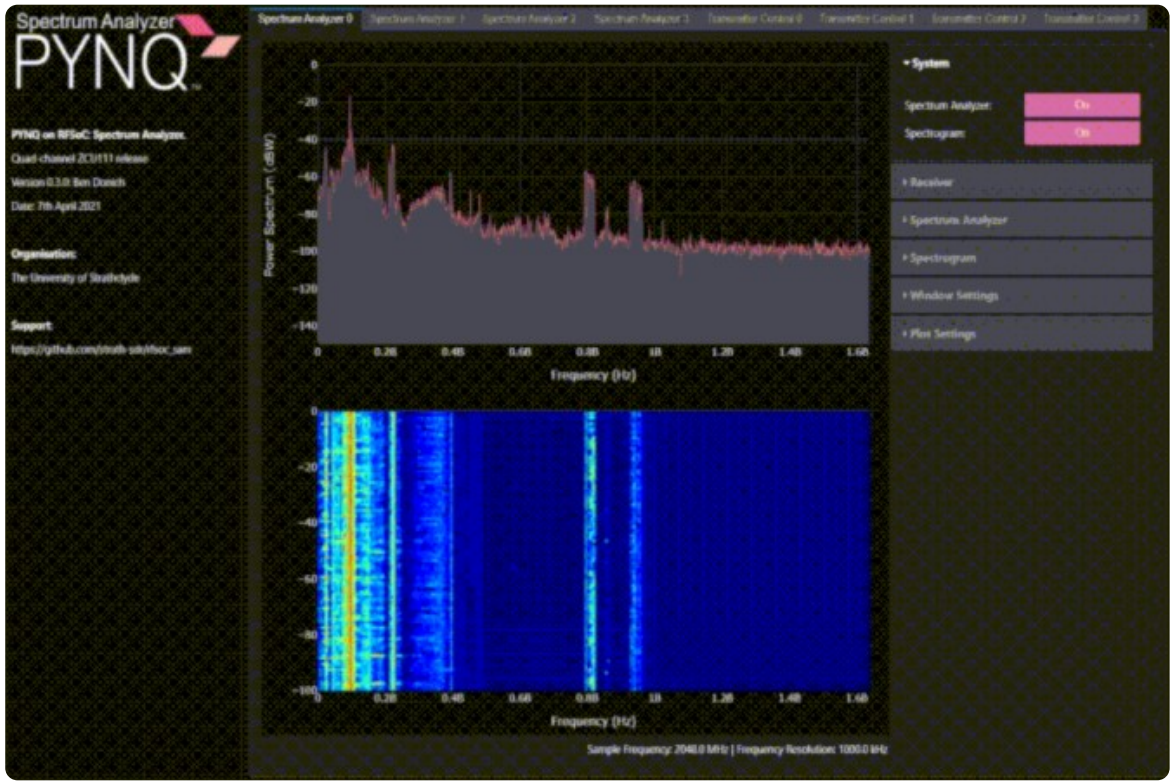
www.rfsocbook.com







Open source examples

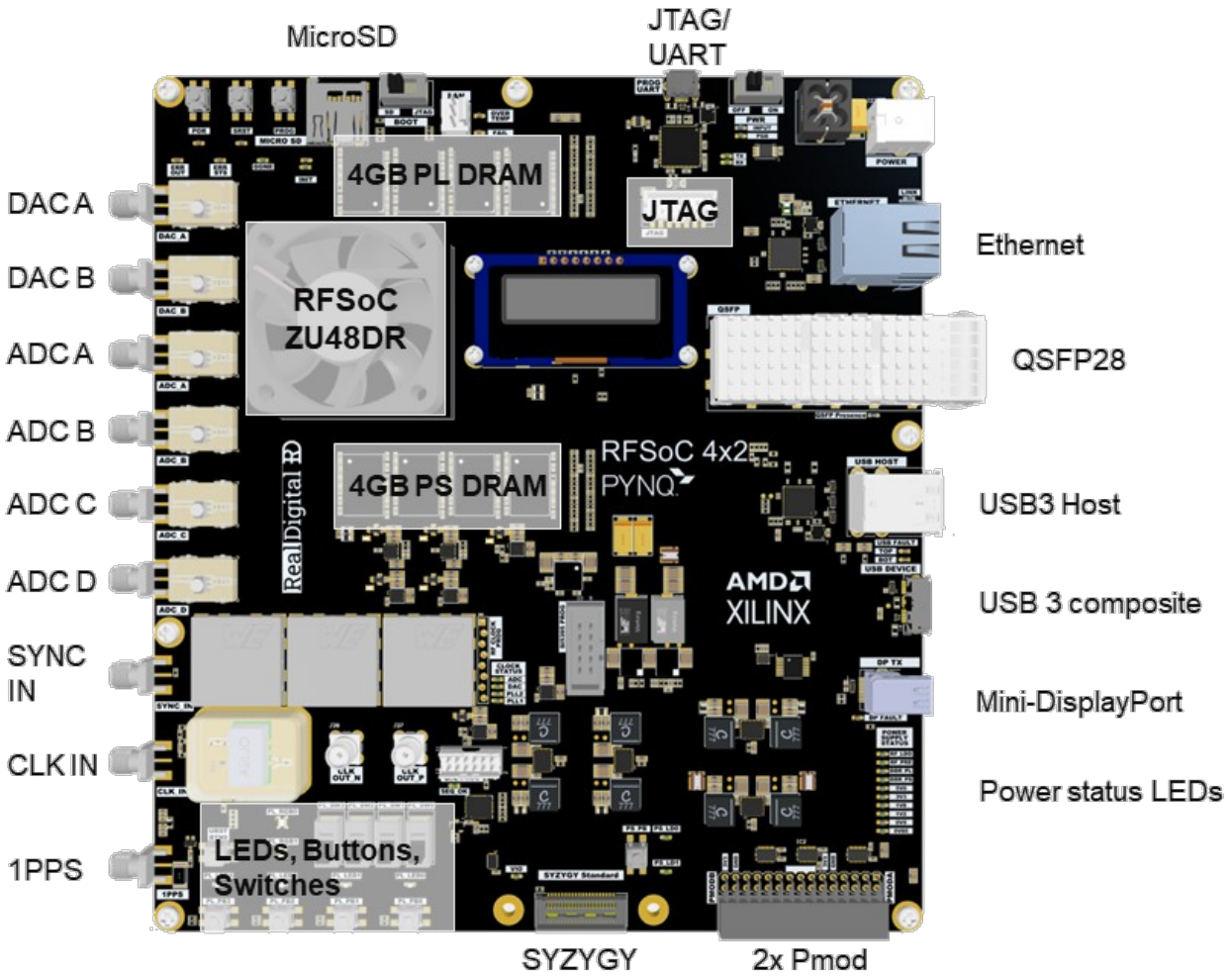


Open Source PYNQ Spectrum Analyzer
https://github.com/strath-sdr/rfsoc_sam

Open Source RFSoC OFDM Transceiver
https://github.com/strath-sdr/rfsoc_ofdm

RFSoC 4x2 platform

- RF Digital to Analog Converters (RF-DACs)
  - 2x
  - 14-bit
  - 9.85 GSPS
- RF Analog to Digital Converters (RF-ADCs)
  - 4x
  - 14-bit
  - 5 GSPS
- 100Gbit/s QSFP network interface
- 64-bit Quad core Arm CPU running PYNQ Linux
- Programmable Logic (PL / FPGA)
- Supported by PYNQ open-source framework with Python & Jupyter notebooks



RFSoC 4x2 http://www.rfsoc-pynq.io/rfsoc_4x2_overview.html

RFSoC offload

StrathSDR

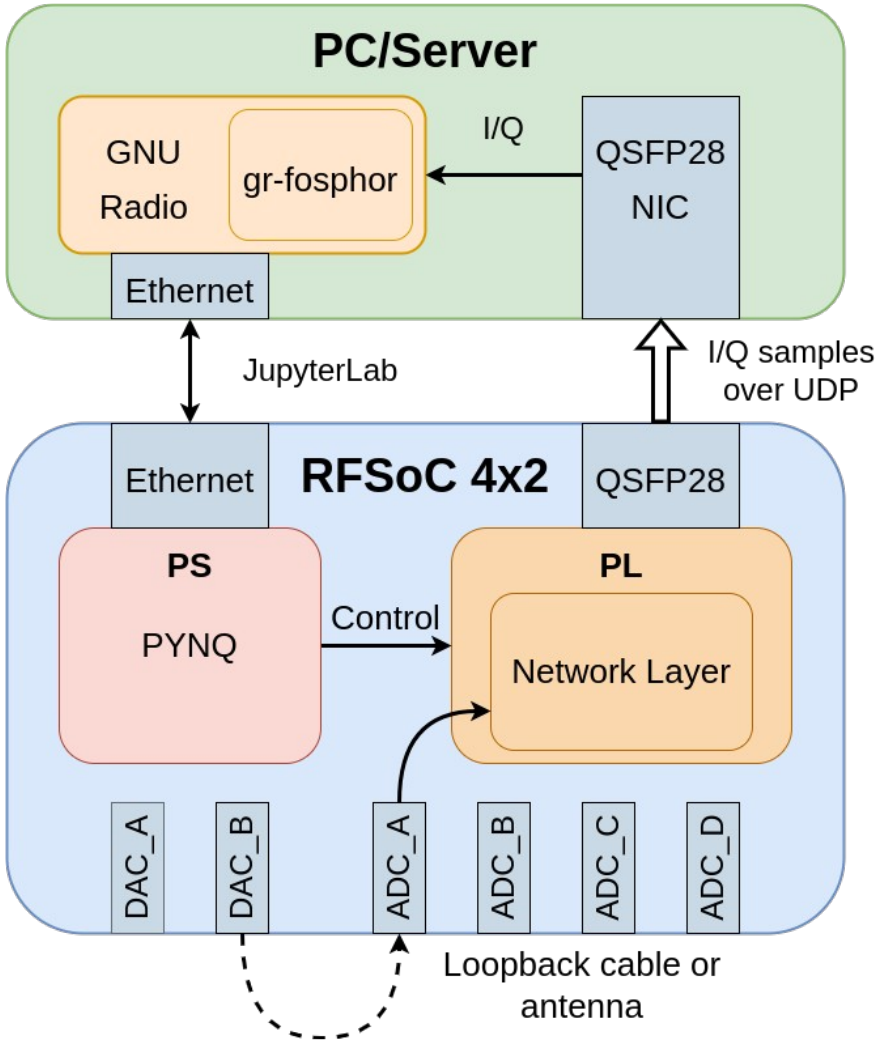
- Modern Software Defined Radio (SDR) systems are capable of multi-gigabit-per-second sampling rates, generating massive amounts of digitized RF data.

- It is often necessary to offload the collected data to a robust server for processing and analysis
  - Spectrum monitoring
  - Disaggregated radio
  - Machine learning
  - Test equipment

- We present an RFSoC offload design capable of 80GBit/s offload in a 100GBit/s network channel with minimum latency.
  - 0 to 2,457.6MHz & 2,457.6MHz to 4915.2MHz Fc
  - Runtime configurable bandwidth

RFSoC Offload Hardware Setup Overview

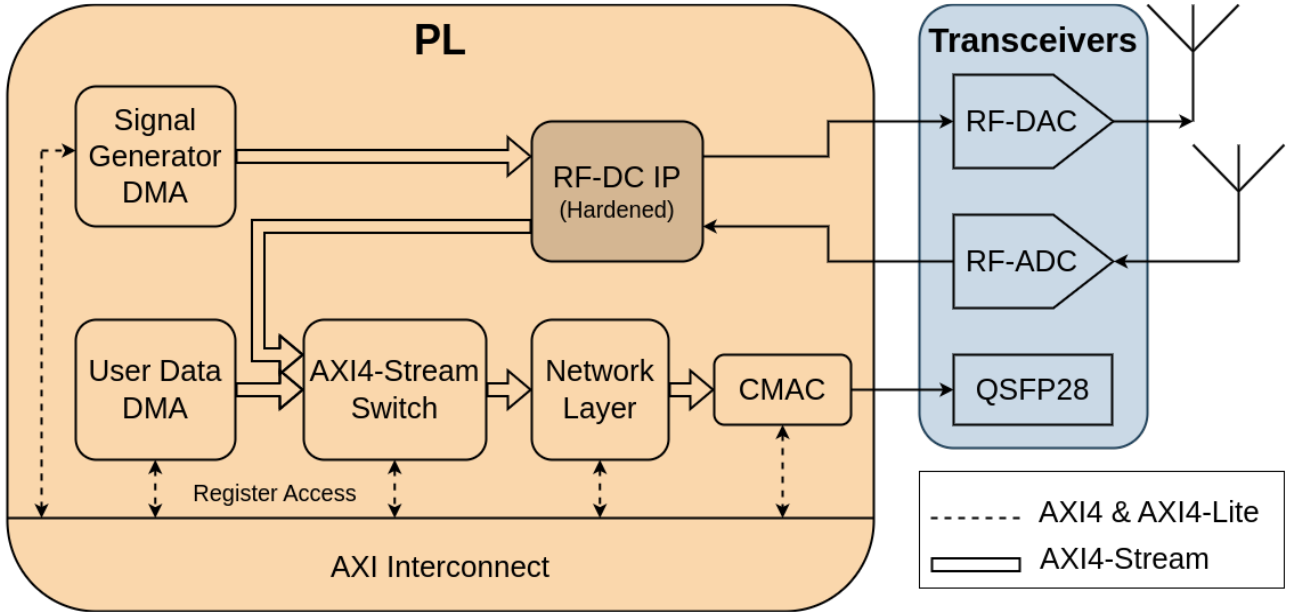
- RFSoC 4x2 board samples RF spectrum from an antenna or via a loopback cable.
- I/Q samples are sent over a 100Gbit network to a PC equipped with a QSFP Network Interface Card (NIC).
- GNU Radio with gr-fosphor is utilized for Graphics Processing Unit (GPU) accelerated, real-time spectrum plotting.
- An Ethernet connection is used to access a JupyterLab interface running on the board and remotely control the RF-DC parameters.



Design overview diagram.

Hardware Architecture

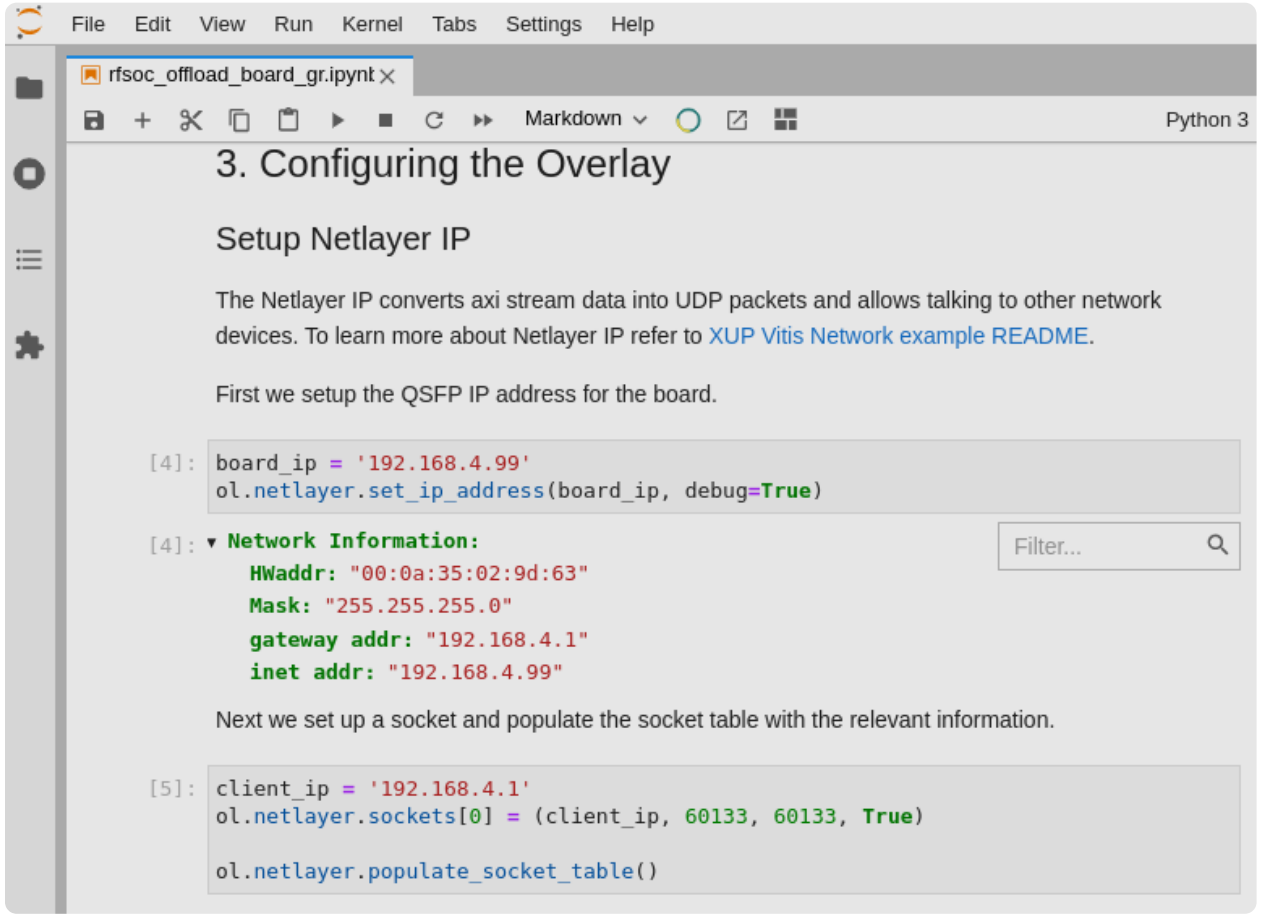
- The high-speed network architecture is implemented entirely on the Programmable Logic (PL), enabling maximum network bandwidth and minimum latency.
- Arbitrary test signals can be generated using Python and transmitted via the RF-DAC using the Signal Generator module.
- Network layer can be driven by either the User Data Direct Memory Access (DMA) or RF-ADC source.
- User Datagram Protocol (UDP) is used for data transfer over the QSFP28 network.



High level overview of the hardware system.

Software Architecture - Board

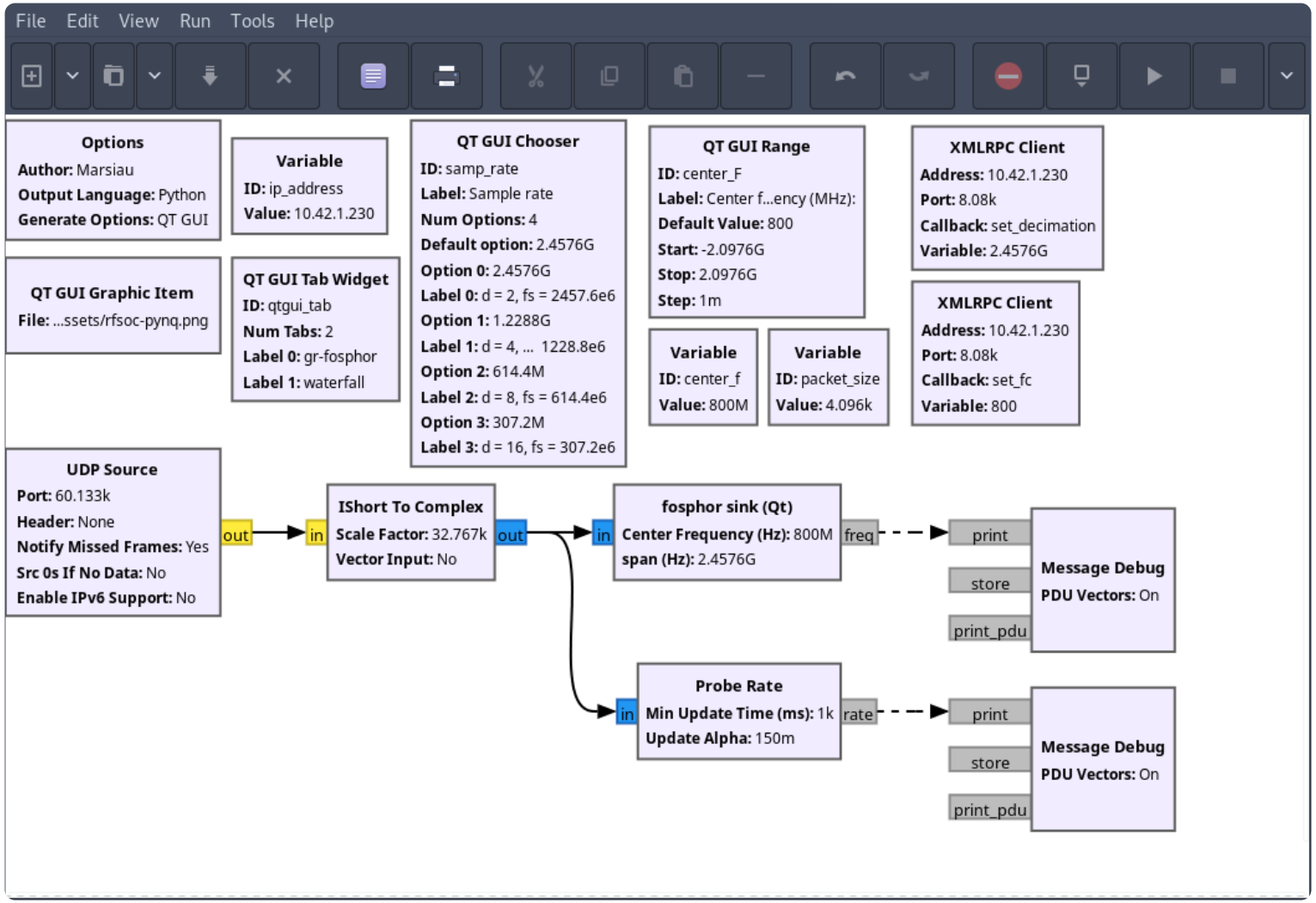
- JupyterLab provides an interactive Python session for dynamic adjustment of software and hardware parameters.
- PYNQ framework provides Python APIs for interacting with the Programmable Logic (PL), controlling parameters of the RF-ADCs, RF-DACs, and programming the internal reference clocks.
- Custom network payloads and signals can be generated in Python and transmitted over the network or through RF-DAC.



JupyterLab interface

Software Architecture – GR Client

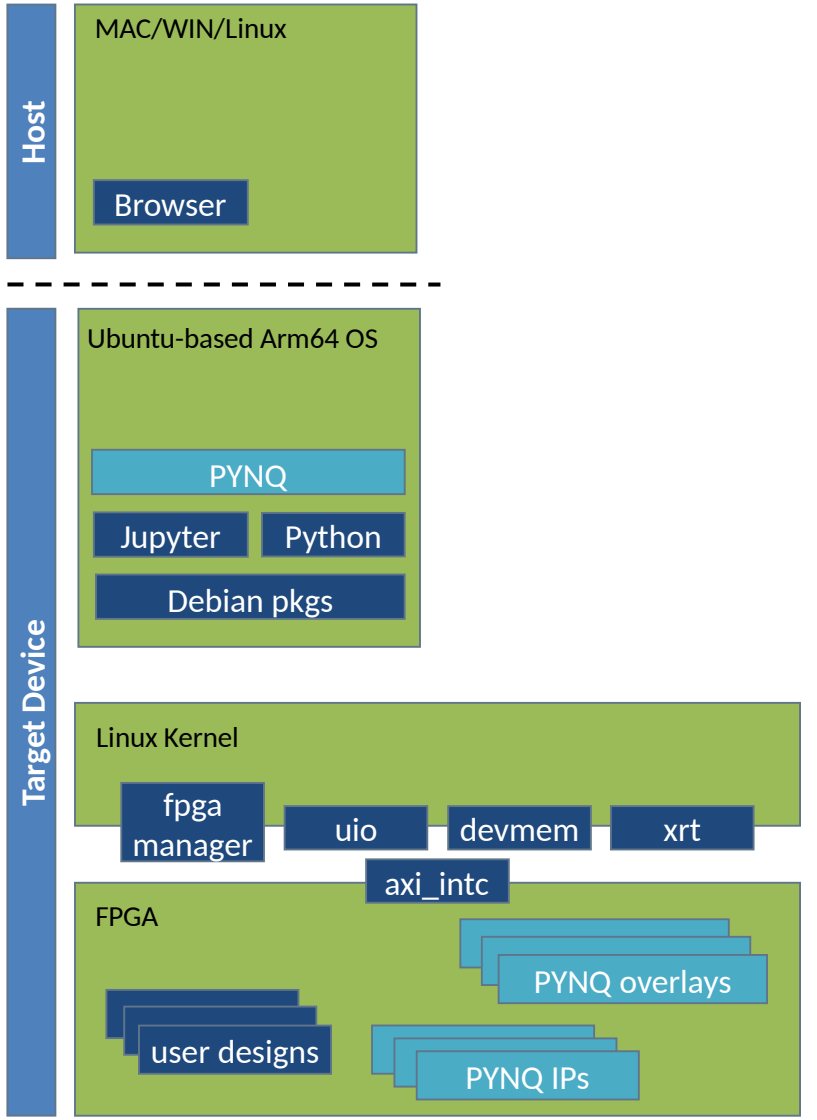
- XMLRPC blocks enable adjustment of remote board parameters.
- Variable Blocks are used to make the design parametric.
- QT GUI Blocks facilitate the user input options.



Full GNU Radio flowgraph

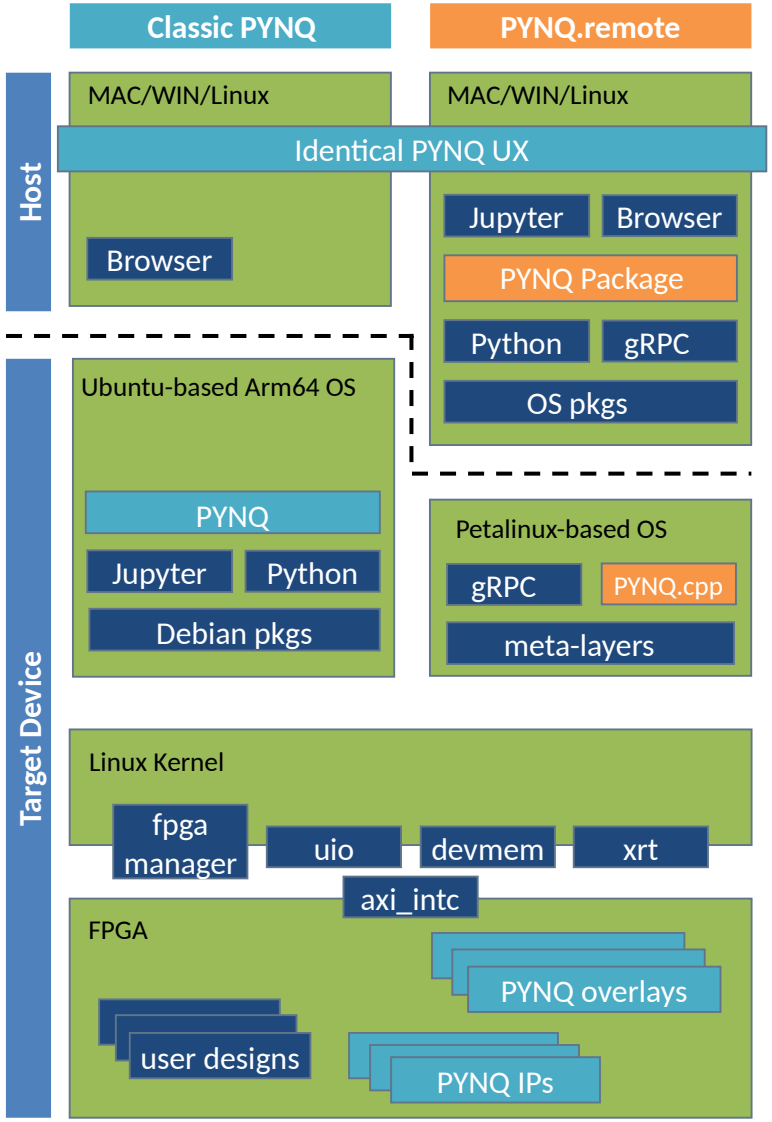
PYNQ: How it works

- Think Raspberry Pi + Programmable Logic
- Users bring their FPGA designs
- PYNQ provides everything you need to get started
  - Ubuntu-based OS image
  - Libraries, drivers, etc.
  - Lots of compatible software
- Run code on-target from a host web browser



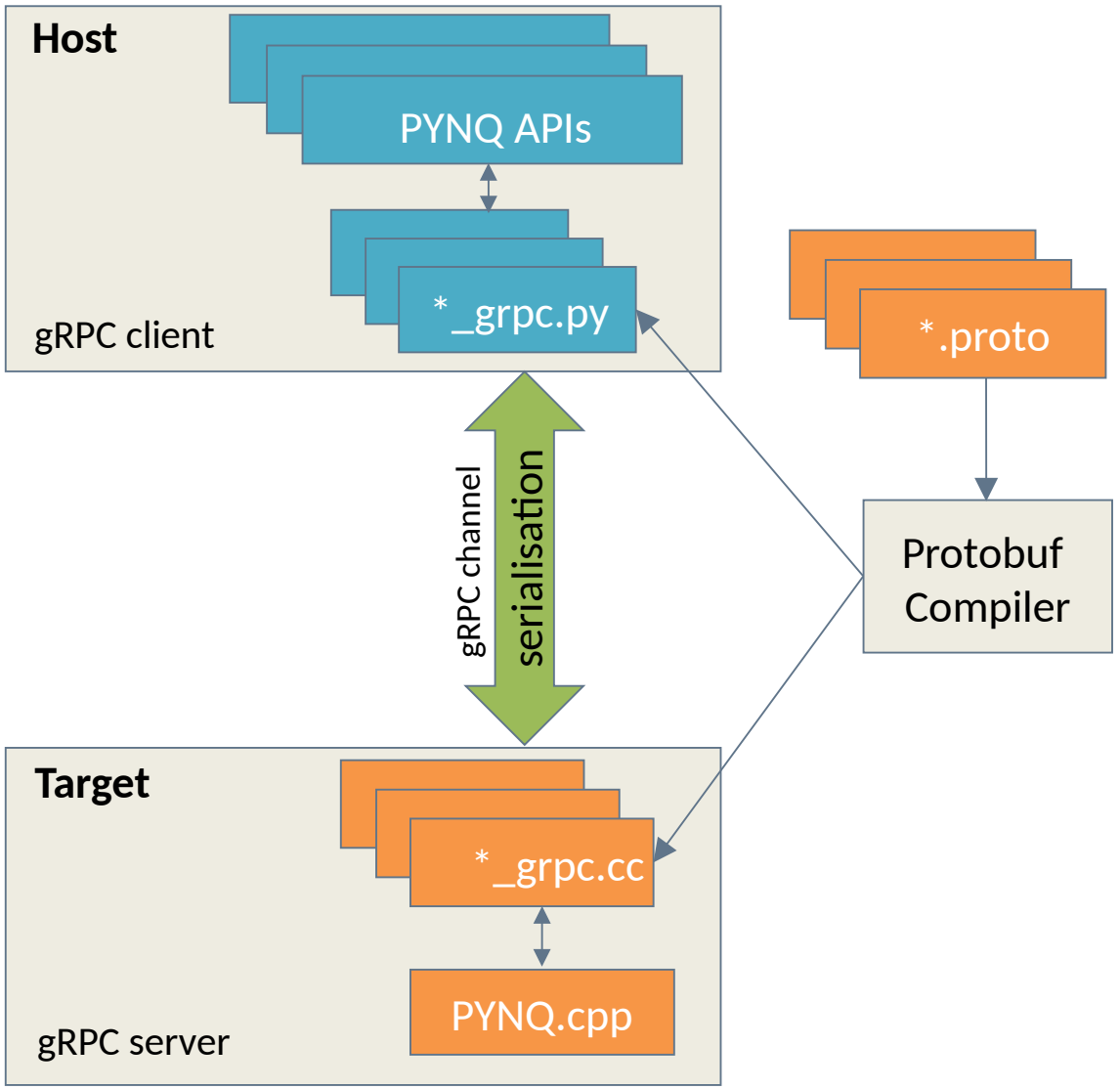
Introducing PYNQ.remote

- Minimal Petalinux image
- PYNQ low-level functionality in C++
  - PYNQ.cpp
- Python API on the host
- User interface remains the same!

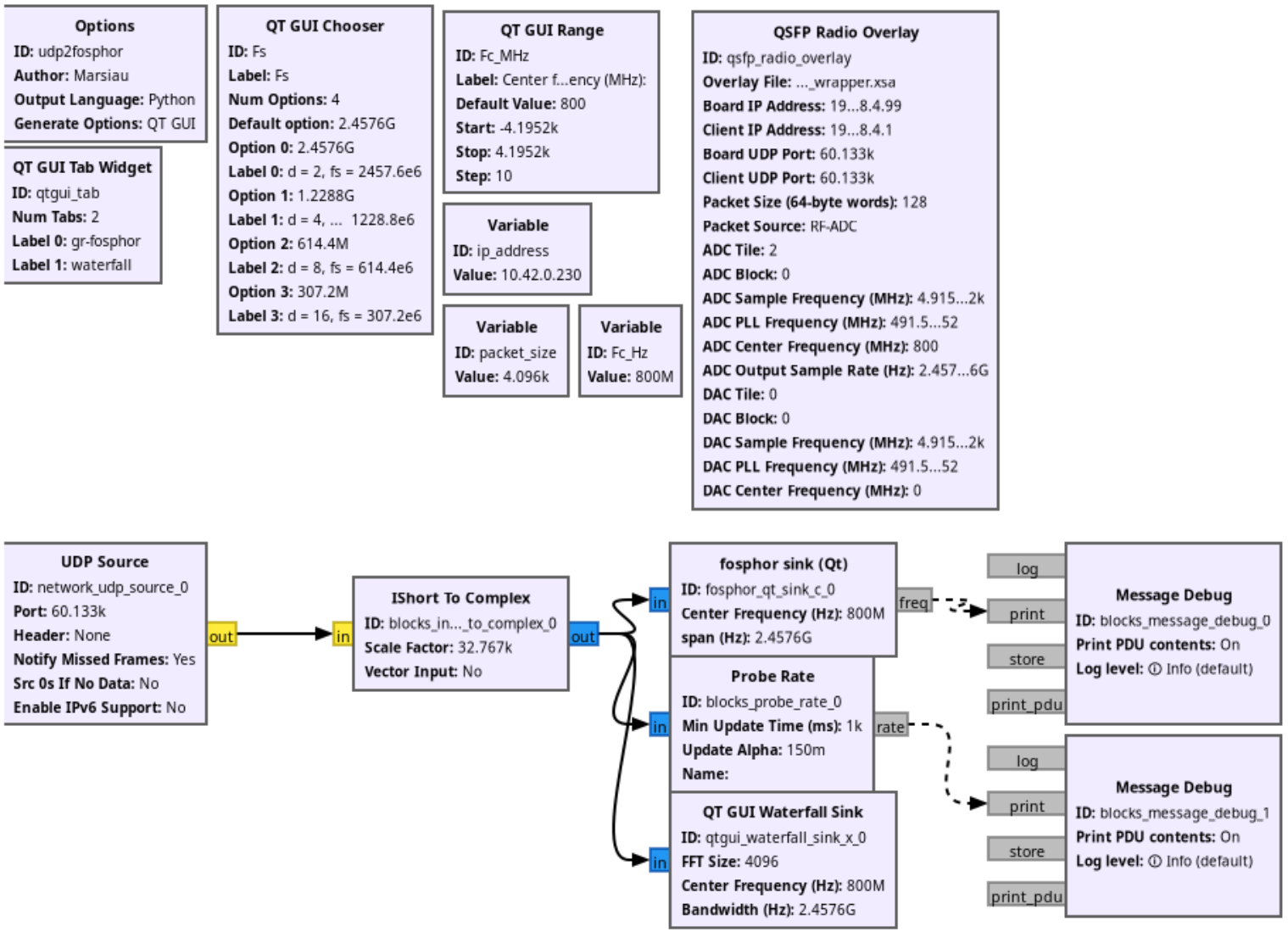


How PYNQ.remote Works: Protocol Buffers

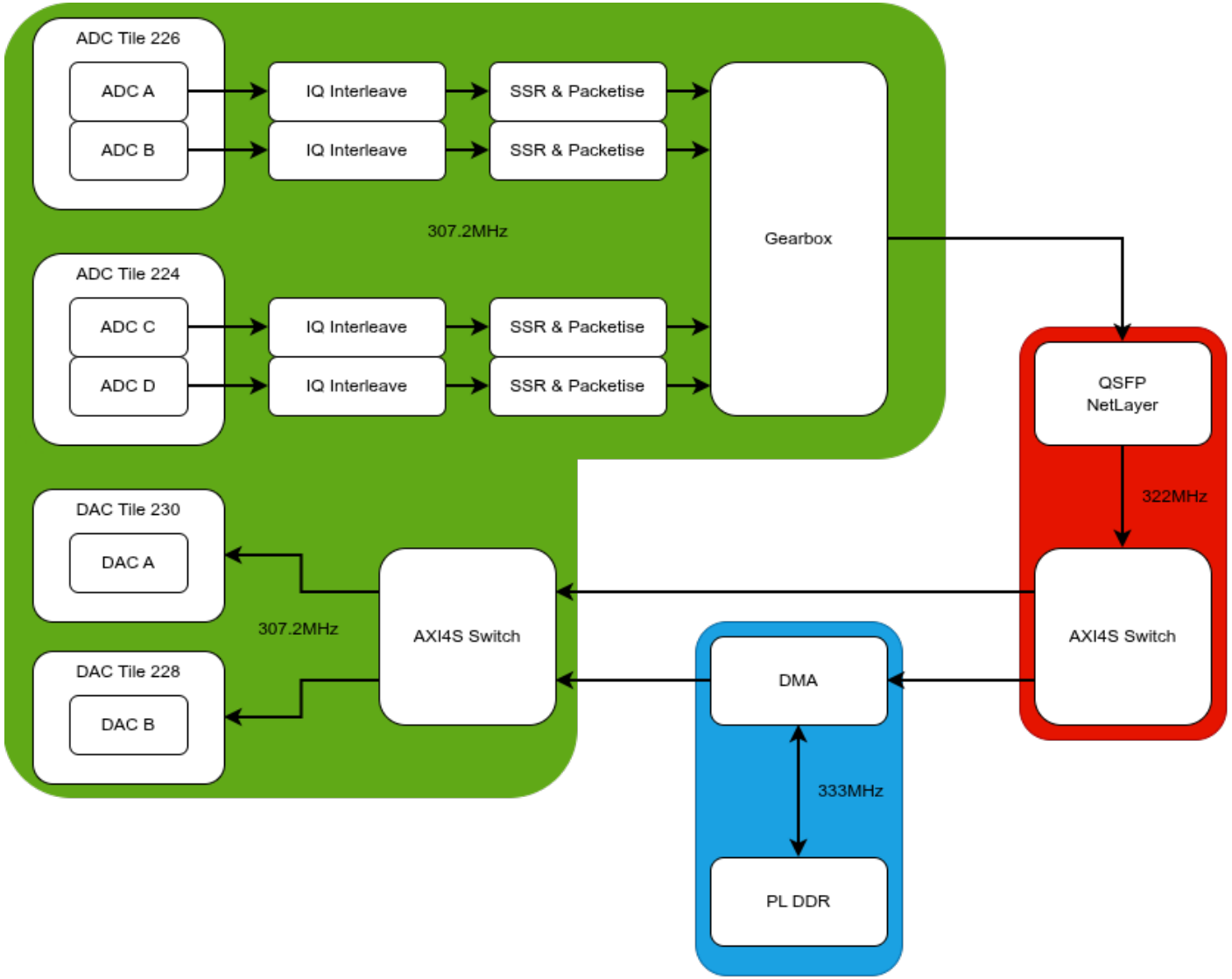
- Compact serialization format
  - think JSON but smaller + faster
- Define schemas for structured data
- Protobuf compiler provides cross-language support
- PYNQ.remote and PYNQ.cpp connect the APIs



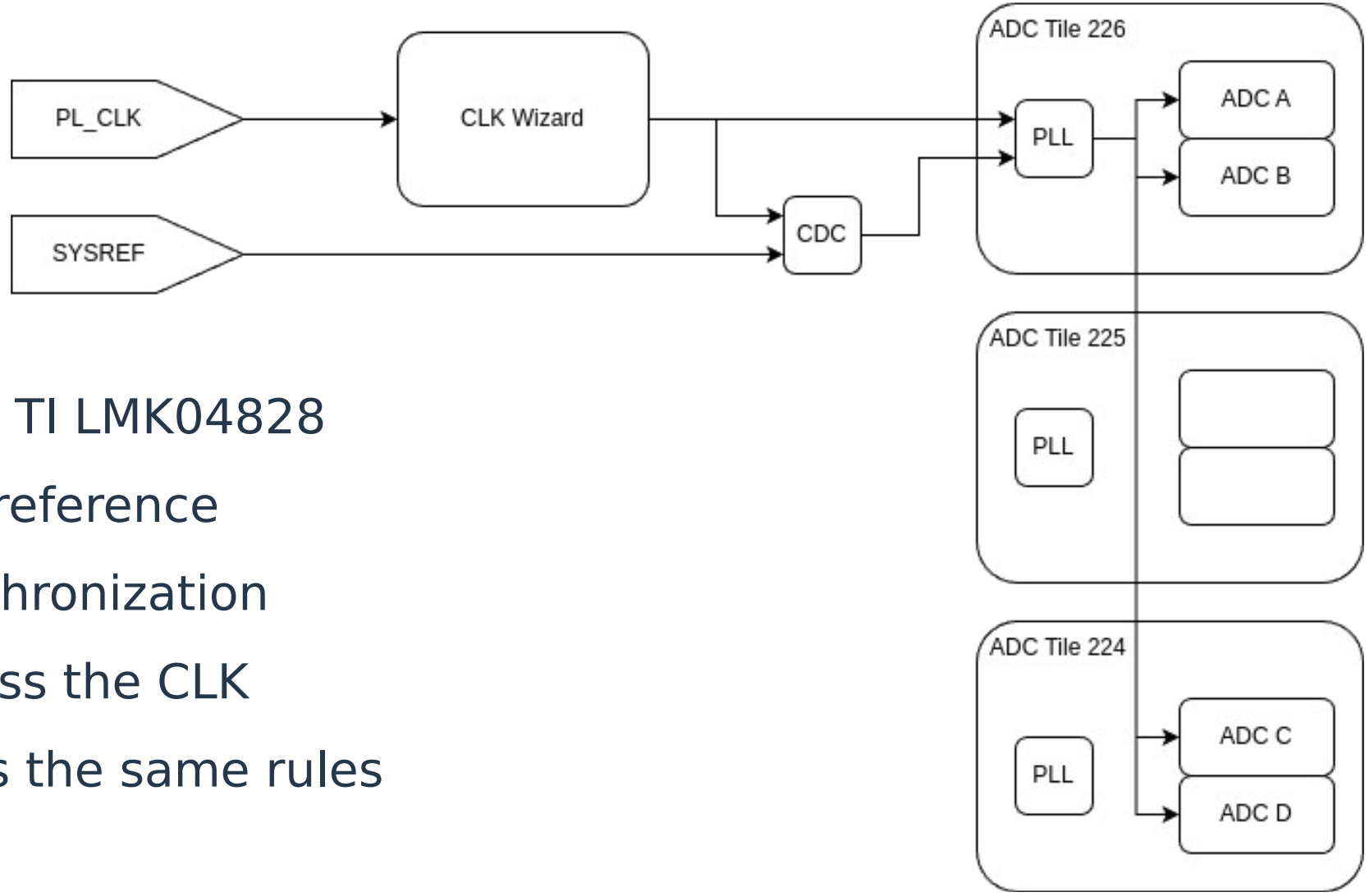
Updated GR flowgraph



Updated Hardware Architecture



Multi Tile Sync (MTS)



- PL_CLK & SYSREF from TI LMK04828
- PL_CLK is used as PLL reference
- SYSREF for phase synchronization
- Tile 225 required to pass the CLK
- DAC subsystem follows the same rules

Network Throughput Test Results

StrathSDR

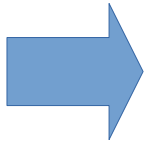
Decimation	Bandwidth (MHz)	RF-ADC Data Output Rate (GBit/s)	Measured Avg. Throughput (GBit/s)
2	**2,457.6**	**78.6432**	**73.40**
4	1,228.8	39.3216	36.81
8	614.4	19.6608	18.41
16	307.2	9.8304	9.20

- To maximize network throughput, Jumbo Frames (up to 9000 bytes of payload) are enabled in the network.
- Experimental results closely align with the calculated ideal throughput.

- RF-ADC data output rate is calculated using:
- $R_{RF-ADC} = B \times C \times Q$
  - B is the bandwidth (RF-ADC sampling rate)
  - C is channels (2 for complex data)
  - Q is the RF-ADC resolution (14 bits with 2-bit padding, total 16 bits)

Additional rates

Decimation	Bandwidth (MHz)	RF-ADC Data Output Rate (GBit/s)
2	2,457.6	78.643
4	1,228.8	39.322
8	614.4	19.661
16	307.2	9.83



DDC	Fs(MHz)	x1 Gbits/s	x2 Gbits/s	x4 Gbits/s
1	4915.2	~~146.48~~	~~292.97~~	~~585.94~~
2	2457.6	78.64	~~157.29~~	~~314.57~~
4	1228.8	39.32	78.64	~~157.29~~
6	819.2	26.21	52.43	~~104.86~~
8	614.4	19.66	39.32	78.64
10	491.52	15.73	31.46	62.91
12	409.6	13.11	26.21	52.43
16	307.2	9.83	19.66	39.32
20	245.76	7.86	15.73	31.46
24	204.8	6.55	13.11	26.21
40	122.88	3.93	7.86	15.73

Conclusions

StrathSDR

73.40 Gbit/s

Measured average throughput at 2,457.6 MHz bandwidth

100 GbE offload connects RFSoC4x2 to GNU Radio for real-time wideband processing.

The updated architecture combines four ADC channels with a network-fed transmit path and selectable decimation rates.

PYNQ.remote moves the Python API to the host, using gRPC and PYNQ.cpp while retaining the PYNQ user interface.

StrathSDR

Thanks for listening!
Engage with us:

https://sdr.eee.strath.ac.uk

@strathSDR

github.com/strath-sdr

github.com/strath-sdr/rfsoc_qsfp_offload