Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

QICK Overview

Leandro Stefanazzi and the QICK Team

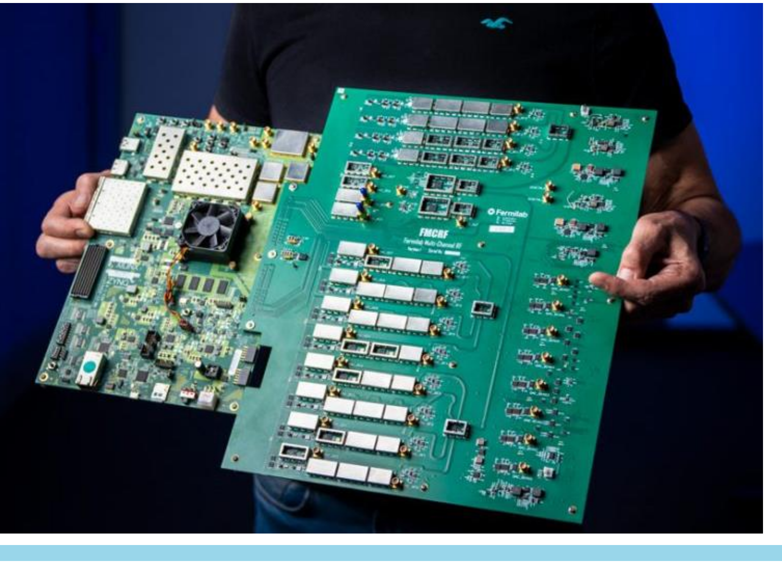
January 12, 2023

QICK

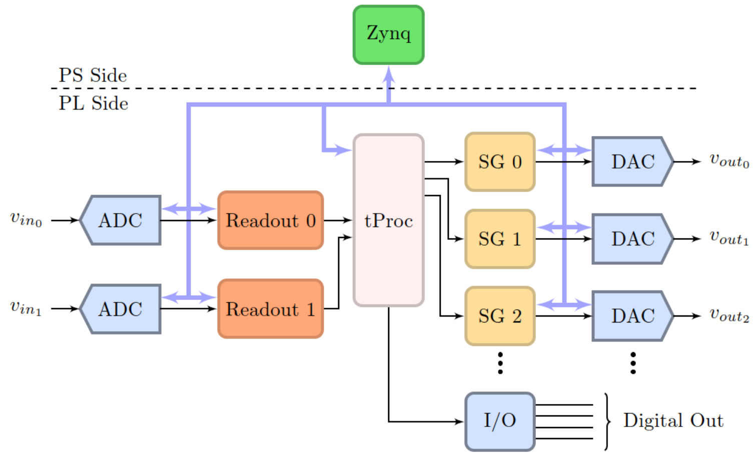
QICK: Quantum Instrumentation Control Kit

What is the QICK?

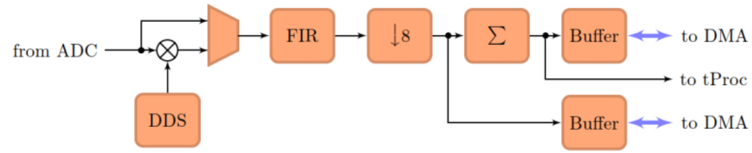
- A flexible system to control and readout qubits.
- Sends RF pulses with fast DACs.
- Reads RF pulses with fast ADCs.
- Allows looping, branching, conditions, etc.
- Provides precise and deterministic timing control.
- FPGA-based: easy to target different applications.

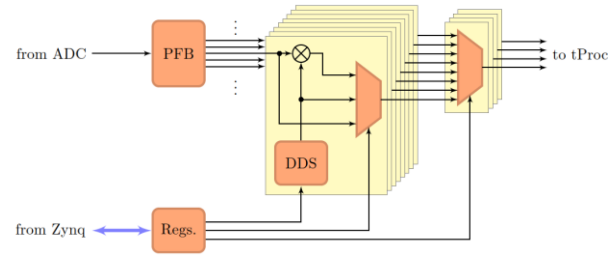


QICK Firmware

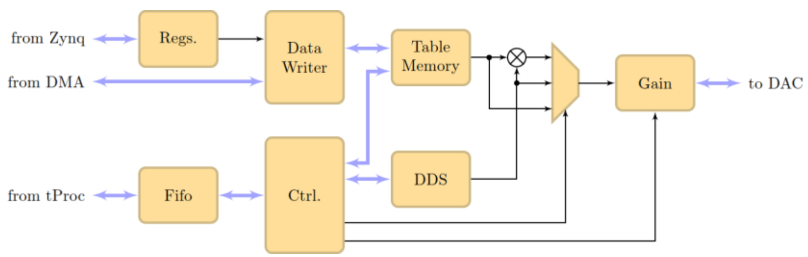


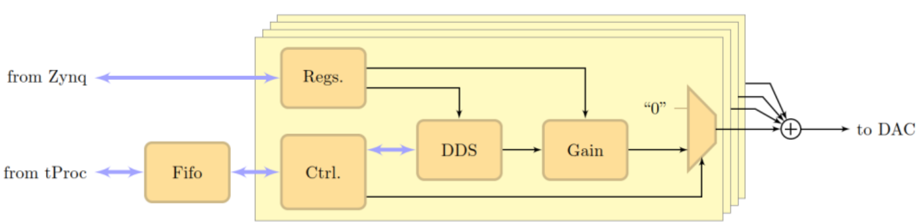
Readout Blocks

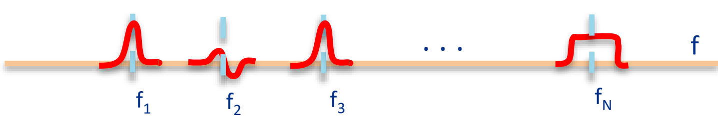




Signal Generators







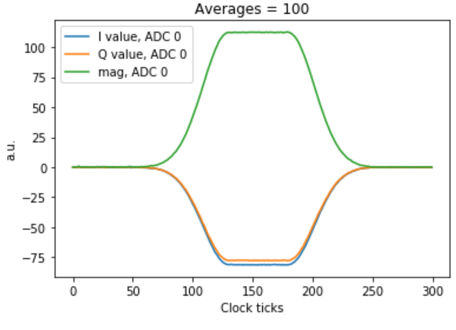
QICK Software

In [8]:

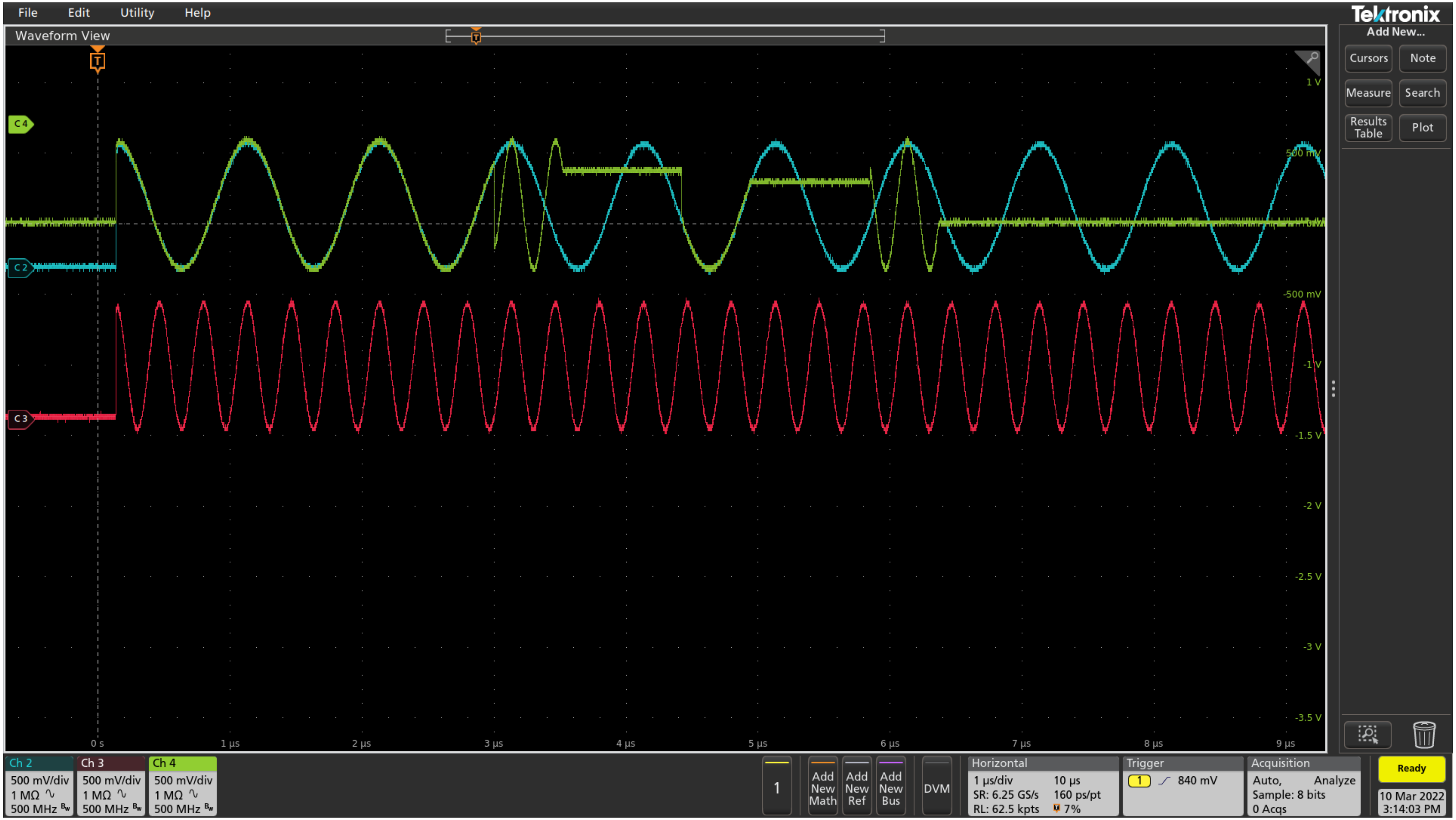
```python
class DoublePulseProgram(AveragerProgram):
    def initialize(self):
        cfg=self.cfg
        res_ch = cfg["res_ch"]

        # set the nyquist zone
        self.declare_gen(ch=cfg["res_ch"], nqz=1)

        # configure the readout lengths and downconversion frequencies (ensuring it is an available
        for ch in cfg["ro_chs"]:
            self.declare_readout(ch=ch, length=self.cfg["readout_length"],
                                 freq=self.cfg["pulse_freq"], gen_ch=cfg["res_ch"])

        # convert frequency to DAC frequency (ensuring it is an available ADC frequency)
        freq = self.freq2reg(cfg["pulse_freq"],gen_ch=res_ch, ro_ch=cfg["ro_chs"][0])
        gain = cfg["pulse_gain"]

        style=self.cfg["pulse_style"]

        if style in ["flat_top","arb"]:
            sigma = cfg["sigma"]
            self.add_gauss(ch=res_ch, name="measure", sigma=sigma, length=sigma*5)

        if style == "const":
            self.default_pulse_registers(ch=res_ch, style=style, freq=freq, gain=gain,
                                         length=cfg["length"])
        elif style == "flat_top":
            # The first half of the waveform ramps up the pulse, the second half ramps down the puls
            self.default_pulse_registers(ch=res_ch, style=style, freq=freq, gain=gain,
                                         waveform="measure", length=cfg["length"])
        elif style == "arb":
            self.default_pulse_registers(ch=res_ch, style=style, freq=freq, gain=gain,
                                         waveform="measure")

        self.synci(200)  # give processor some time to configure pulses

    def body(self):
        phase1 = self.deg2reg(self.cfg["res_phase"], gen_ch=self.cfg["res_ch"])
        phase2 = self.deg2reg(self.cfg["res_phase"]+90, gen_ch=self.cfg["res_ch"])
        # fire a single trigger, but two pulses offset by 200 tProc clock ticks
        # with the first ADC trigger, pulse PMOD0_0 for a scope trigger
        # after the full sequence is set up, pause the tProc until readout is done
        # and increment the time counter to give some time before the next measurement
        # (the syncdelay also lets the tProc get back ahead of the clock)
        self.trigger(adcs=self.ro_chs,
                     pins=[0],
                     adc_trig_offset=self.cfg["adc_trig_offset"])
        self.set_pulse_registers(ch=self.cfg["res_ch"], phase=phase1)
        self.pulse(ch=self.cfg["res_ch"], t=0)
        self.set_pulse_registers(ch=self.cfg["res_ch"], phase=phase2)
        self.pulse(ch=self.cfg["res_ch"], t=200)
        self.wait_all()
        self.sync_all(self.us2cycles(self.cfg["relax_delay"]))
```

```python
config={"res_ch":6, # --Fixed
        "ro_chs":[0], # --Fixed
        "reps":1, # --Fixed
        "relax_delay":1.0, # --us
        "res_phase":0, # --degrees
        "pulse_style": "flat_top", # --Fixed
        "length": 50, # [Clock ticks]
        # Try varying  length from 10-100 clock ticks
        "sigma": 30, # [Clock ticks]
        # Try varying sigma from 10-50 clock ticks

        "readout_length":300, # [Clock ticks]
        # Try varying readout_length from 50-1000 clock ticks

        "pulse_gain":5000, # [DAC units]
        # Try varying pulse_gain from 500 to 30000 DAC units

        "pulse_freq": 100, # [MHz]
        # In this program the signal is up and downconverted digitally so you won't see any frequency
        # components in the I/Q traces below. But since the signal gain depends on frequency,
        # if you lower pulse_freq you will see an increased gain.

        "adc_trig_offset": 100, # [Clock ticks]
        # Try varying adc_trig_offset from 100 to 220 clock ticks

        "soft_avgs":100
        # Try varying soft_avgs from 1 to 200 averages

       }

###################
# Try it yourself !
###################

prog =LoopbackProgram(soccfg, config)
iq_list = prog.acquire_decimated(soc, load_pulses=True, progress=True, debug=False)
```

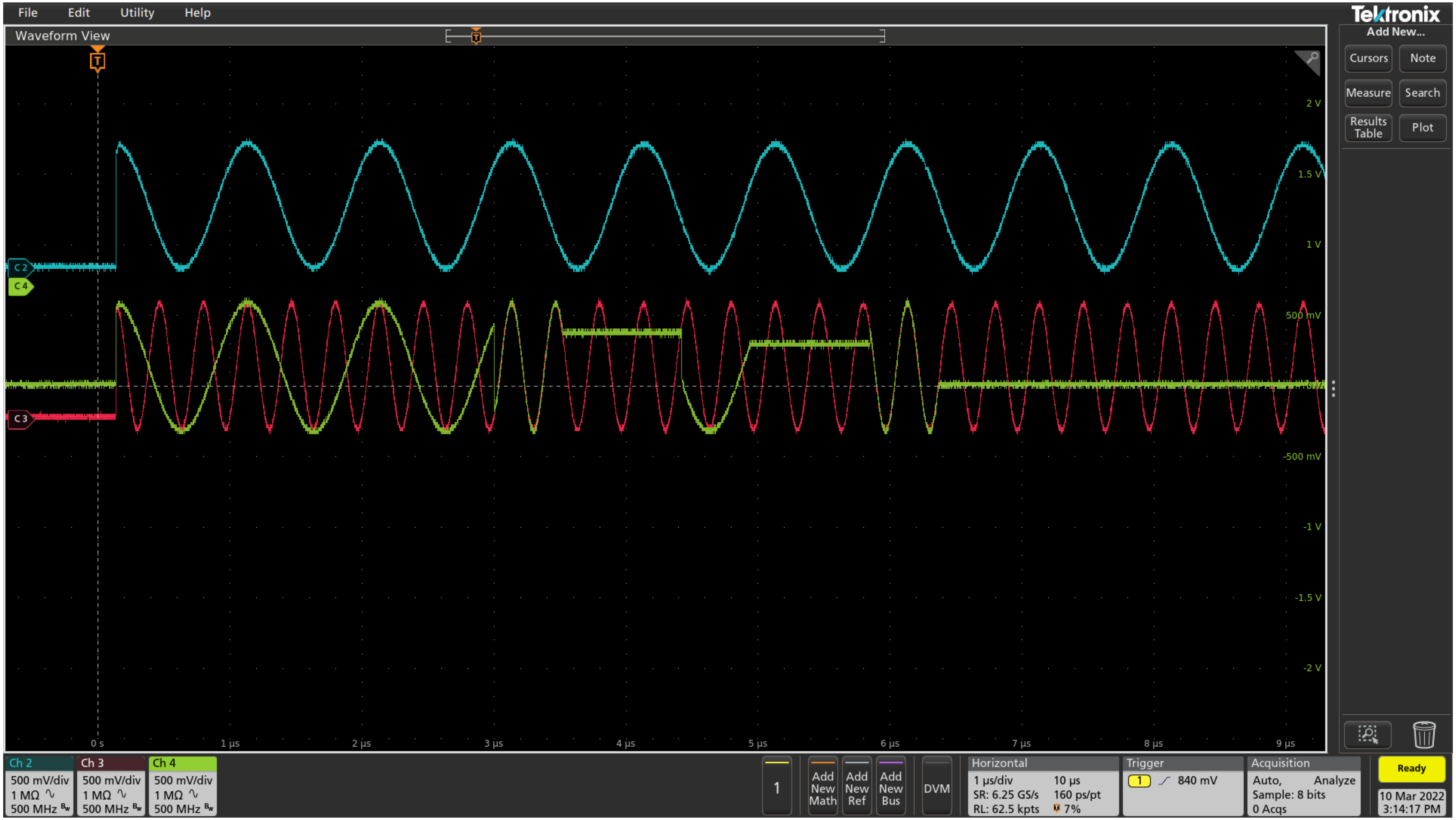
```python
# Plot results.
plt.figure(1)
for ii, iq in enumerate(iq_list):
    plt.plot(iq[0], label="I value, ADC %d"%(config['ro_chs'][ii]))
    plt.plot(iq[1], label="Q value, ADC %d"%(config['ro_chs'][ii]))
    plt.plot(np.abs(iq[0]+1j*iq[1]), label="mag, ADC %d"%(config['ro_chs'][ii]))
plt.ylabel("a.u.")
plt.xlabel("Clock ticks")
plt.title("Averages = " + str(config["soft_avgs"]))
plt.legend()
plt.savefig("images/Send_recieve_pulse_flattop.pdf", dpi=350)
```



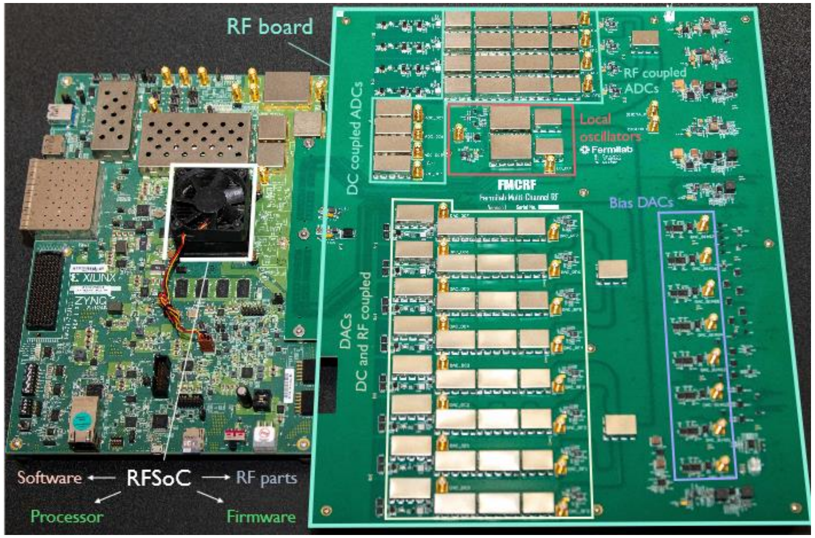
Phase Control 1

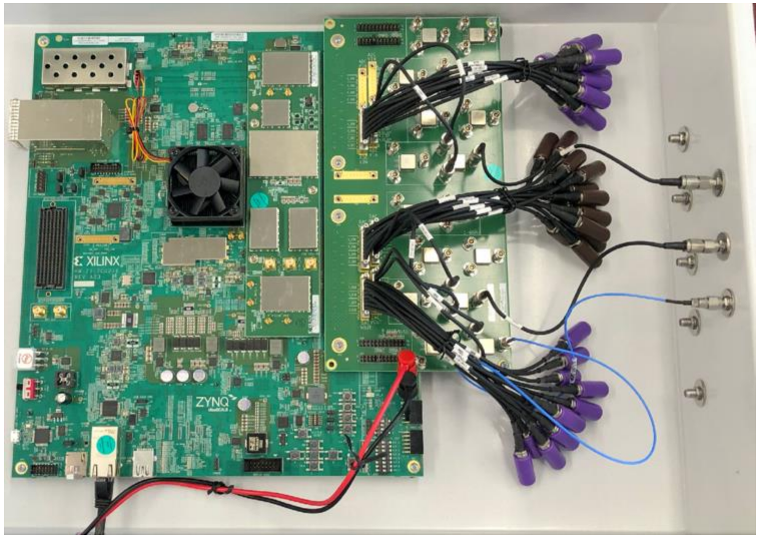


Phase Control 2



QICK 1 and 2

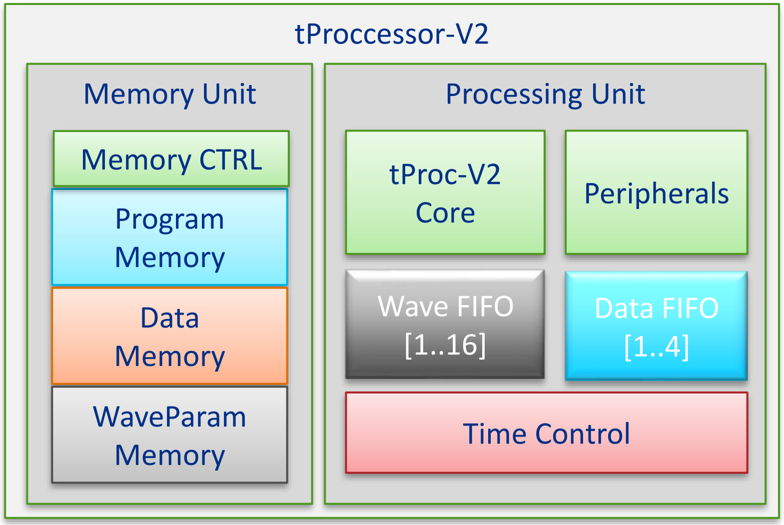




- **Based on ZCU111.**
- **Companion RF board.**
- **8 RF or fast DC-1.5 GHz outputs.**
- **4 RF inputs, 4 DC-1.5 GHz inputs.**
- **8 biasing DACs, 20-bit.**

- **Based on ZCU216.**
- **Companion RF board being designed.**
- **16 RF or fast DC-1.5 GHz outputs.**
- **4 RF inputs, 4 DC-1.5 GHz inputs.**
- **8 biasing DACs, 20-bit.**

Coming soon 1: tProcessor Version 2



Features

- 5-stage pipeline architecture.
- Nested function CALL/RETURN.
- Dedicated Wave Memory.

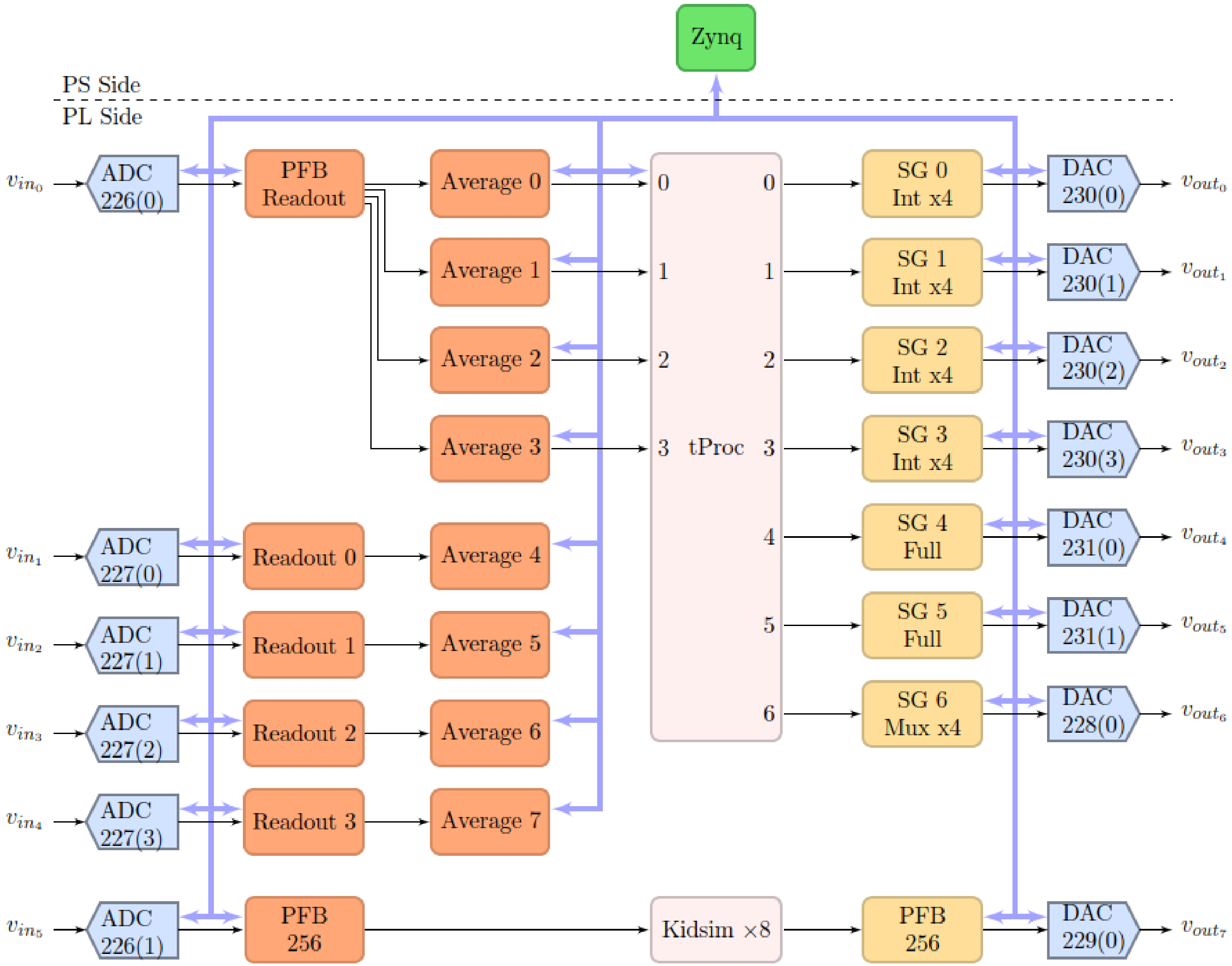
Peripherals

- Dedicated Multiplication and Division Units.
- Pseudo Random Number Generator.

Ports

- Configurable input AXIS IFs from 1 to 16.
- Configurable output AXIS IFs from 1 to 16.

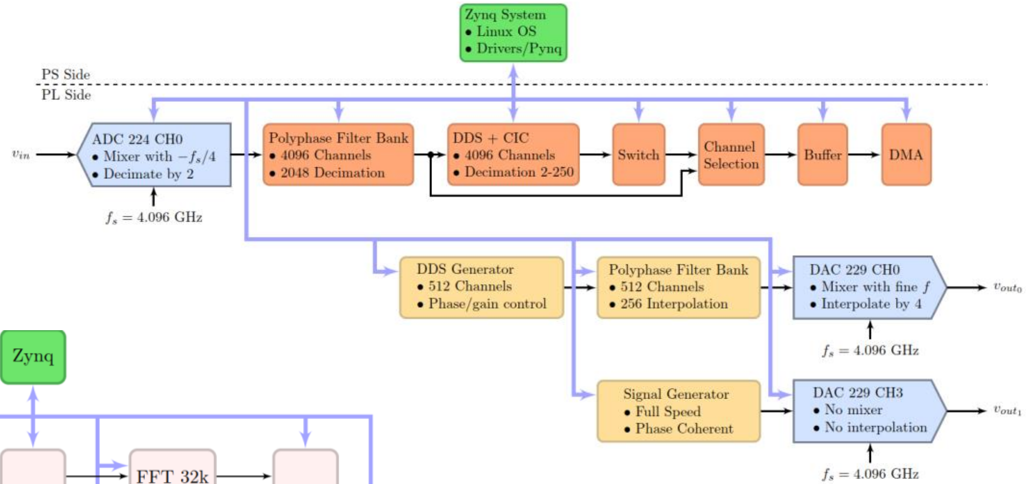
Coming soon 2: QICK for Testing

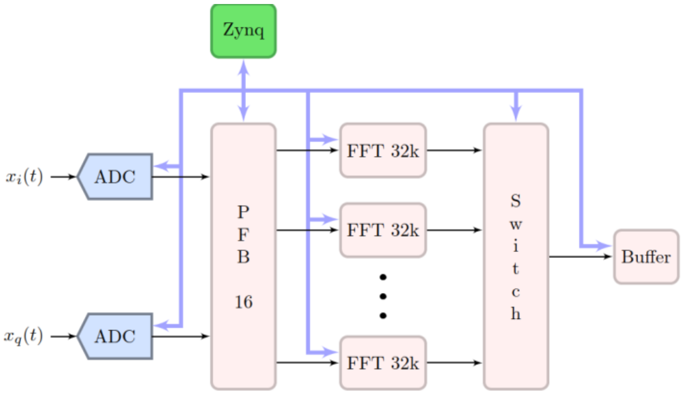


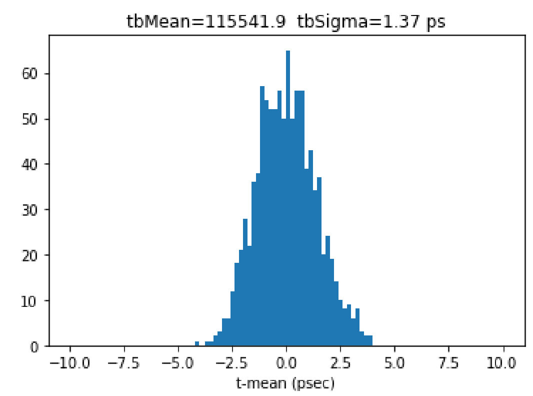
Fermilab

QICK for detectors

- QICK is being used for CMB, dark matter, quantum networks.
- MKIDs for CMB: 1k or more pixels per RF line.
- QN: fast timing measurements with less than 2ps resolution.
- BREAD: 500k points FFT with zero dead-time and averaging.







Fermilab

Summary

- The QICK is an easy-to-use readout and control system.
- QICK is being extended and used in different detectors.
- Python based, which makes developing experiments easy.
- FPGA reconfigurability to target different system needs.
- PYNQ: multiple Firmware images can co-exist.
- Accessible Git Hub repository and open-source philosophy.
- NEW tProcessor will add functionality and speed.
- NEW modular RF companion board to allow scalability.

Driven by experimenters, for experimenters!!

Thank you!!

Software, firmware, demos:
https://github.com/openquantumhardware/qick

Documentation:
https://qick-docs.readthedocs.io/

Talk to us: #qick on http://discord.unitary.fund/

Reference:
https://arxiv.org/abs/2110.00557

QICK

QICK: Quantum Instrumentation Control Kit