ARTIQ and Sinara: Open Software and Hardware Stacks for Quantum Physics

Robert Jördens[1,2], Sébastien Bourdeauducq[2,1], Peter Zotov[2], Grzegorz Kasprowicz[3,4], Paweł Kulik[3], Daniel Slichter[5], David Allcock[5], Joe Britton[6], Thomas Harty[7], Chris Ballance[7], Ulrich Warring[8], Christian Ospelkaus[9,10], Nils Huntemann[10], Piet O. Schmidt[10,9]
1. QUARTIQ GmbH Germany, 2. M-Labs Ltd. Hong Kong, 3. Warsaw University of Technology Poland, 4. Creotech Instruments S.A. Poland, 5. National Institute of Standards and Technology USA, 6. University of Maryland USA, 7. University of Oxford United Kingdom, 8. Universität Freiburg Germany, 9. Leibniz Universität Hannover Germany, 10. Physikalisch Technische Bundesanstalt Germany

ARTIQ control framework

- **A**dvanced **R**eal-**T**ime **I**nfrastructure for **Q**uantum physics, integrated software/gateware/hardware system that controls atomic physics experiments
- High performance — nanosecond resolution, hundreds of ns latency
- Expressive — describe algorithms with few lines of code
- Portable — treat hardware (FPGA boards) as commodity, make software quickly and reliably deployable
- Modular — separate components as much as possible: managing/scheduling experiments, driving distributed devices, analyzing/displaying/archiving results
- Open — developed with and for research groups worldwide as open source software (LGPLv3+)

Define a simple timing language

```
trigger.sync()                # wait for trigger input
start = now()                 # capture trigger time
for i in range(3):
    delay(5*us)
    dds.pulse(900*MHz, 7*us)  # first pulse 5 µs after trigger
at(start + 1*ms)              # re-reference time-line
dds.pulse(200*MHz, 11*us)     # exactly 1 ms after trigger
```

- Written in a subset of Python
- Executed on a CPU embedded on a FPGA (the *core device*)
- `now()`, `at()`, `delay()` describe time-line of an experiment
- Exact time is kept in an internal variable
- That variable only loosely tracks the execution time of CPU instructions
- The value of that variable is exchanged with the RTIO fabric that does precise timing

Convenient syntax additions

```
with sequential:
    with parallel:
        a.pulse(100*MHz, 10*us)
        b.pulse(200*MHz, 20*us)
    with parallel:
        c.pulse(300*MHz, 30*us)
        d.pulse(400*MHz, 20*us)
```

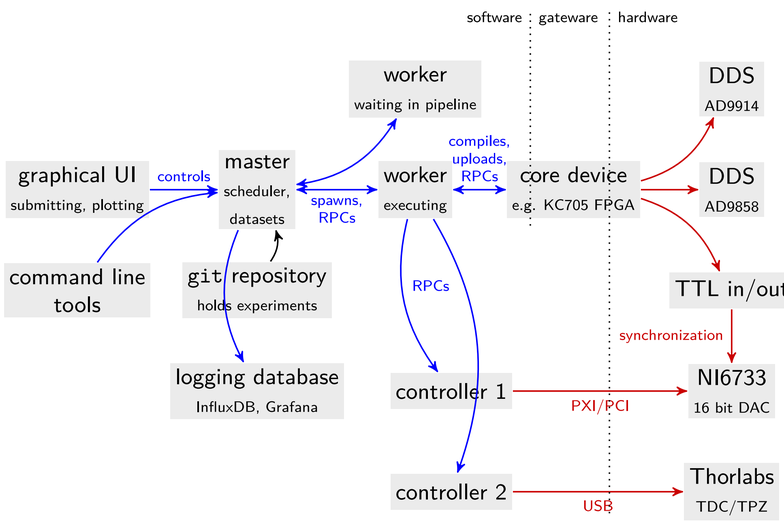
- Experiments are inherently parallel: simultaneous laser pulses, parallel cooling of ions in different trap zones
- `parallel` and `sequential` contexts with arbitrary nesting
- `a` and `b` pulses both start at the same time
- `c` and `d` pulses both start when `a` and `b` are both done (after 20 µs)
- Implemented by inlining, loop-unrolling, and interleaving

RPC to handle distributed non-RT hardware

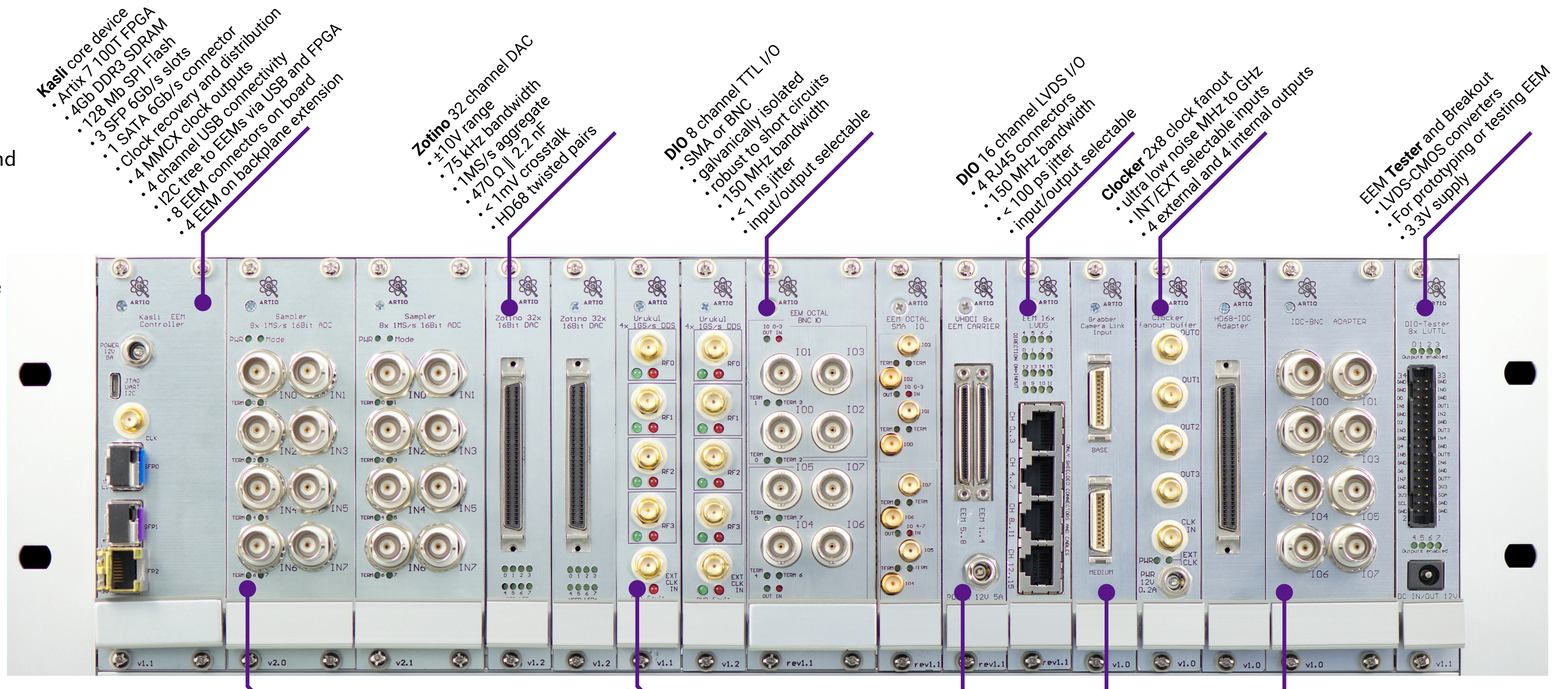
```
class Experiment:
    def prepare(self):                # runs on the host
        self.motor.move_to(20*mm)     # slow RS232 motor controller

    @kernel
    def run(self):                    # runs on the RT core device
        self.prepare()                # converted into an RPC
```

- When a kernel function calls a non-kernel function, it generates a RPC
- The callee is executed on the host
- Mechanism to report results and control slow devices
- The kernel must have a loose real-time constraint (a long `delay`) or means of re-synchronization to cover communication, host, and device delays



Sinara open hardware ecosystem



not shown:

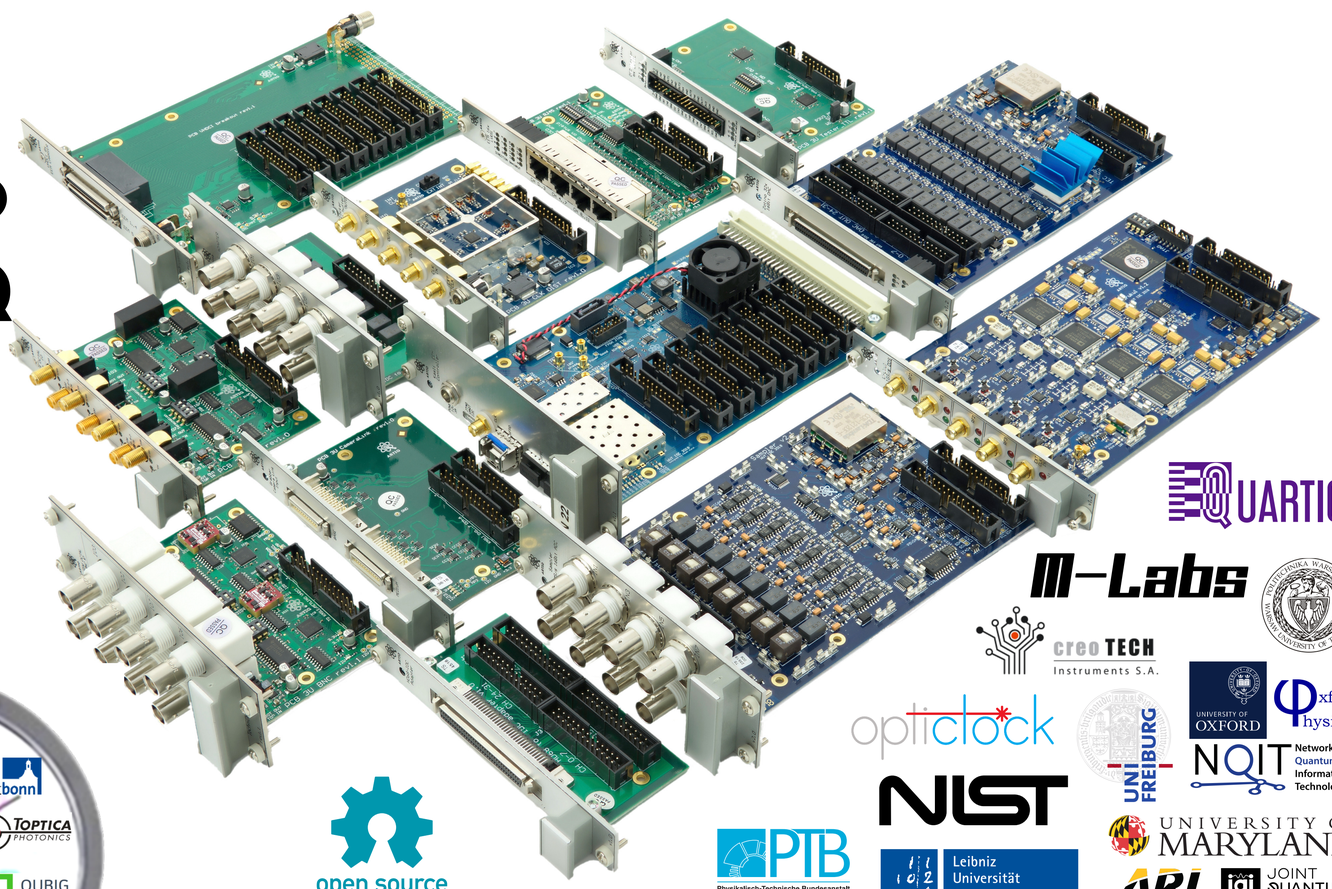
Kasli Backplane extension
- Adds 4 more EEM connectors
- Clock output and power input
- In-line with Kasli

Mirny 4 channel programmable VCO
- 30 MHz to 6/18 GHz
- CPLD for IO and SPI mux
- Analog frontend like Urukul
- Planning phase

Humpback EEM carrier for FPGAs/µC
- Realtime control of custom digital hardware
- Either as a simple "shield" or as a carrier for FPGA/µC development/evaluation/custom boards
- Provides power, mounting, and connectivity
- Planning phase

Booster 8 channel power amplifier
- 40-500 MHz
- P1dB 36 dBm, 40 dB gain
- >35% total power efficiency
- Protection, monitoring
- Remote control

ARTIQ



open source hardware

Sayma multi-channel Smart Arbitrary Waveform Generator (SAWG)

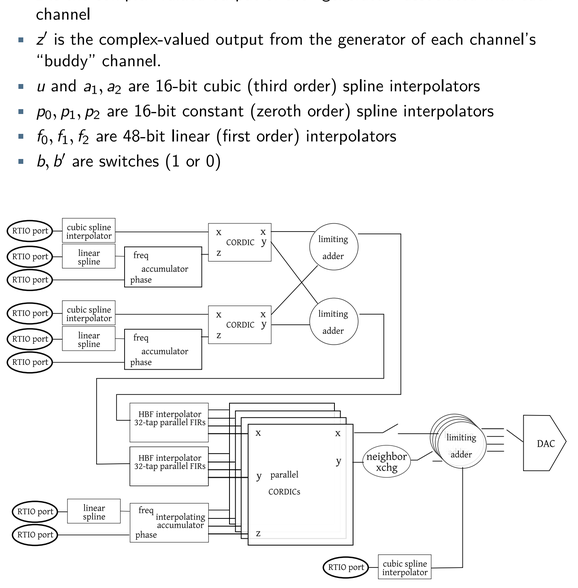
SAWG Parametrization

Each channel emits waveforms of the general parametrization:

$$z = \left(a_1 e^{i(f_1 t + p_1)} + a_2 e^{i(f_2 t + p_2)}\right) e^{i(f_0 t + p_0)}$$

$$o = u + b\,\mathrm{Re}(z) + b'\,\mathrm{Im}(z')$$

- o is the (real valued) output of a channel
- z is the complex-valued output of the "generator" associated with each channel
- z' is the complex-valued output from the generator of each channel's "buddy" channel.
- u and a_1, a_2 are 16-bit cubic (third order) spline interpolators
- p_0, p_1, p_2 are 16-bit constant (zeroth order) spline interpolators
- f_0, f_1, f_2 are 48-bit linear (first order) interpolators
- b, b' are switches (1 or 0)



- µTCA AMC and RTM
- Kintex Ultrascale XCKU040
- 2x AD9154 DAC, JESD204B
- 2x AD9656 ADC, JESD204B
- 8 DAC channels, 2.4 GS/s
- 8 ADC channels, 125 MS/s
- Multiple analog frontends for different bands
- Low noise PLL and flexible clocking tree
- Multiple boards synchronized via DRTIO