Advanced Real-Time Infrastructure for Quantum physics (ARTIQ)

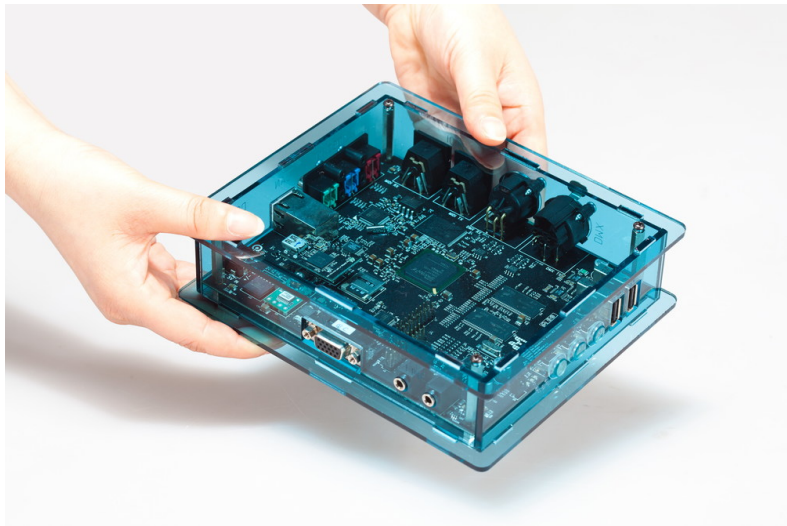
Robert Jördens

M-Labs `http://www.m-labs.hk`

M-Labs Limited

- Founded in 2014
- Incorporated in Hong Kong
- Now 4 full-time staff
- Engineering contracts for physics are fun:
  - Purpose
  - Challenging problems, multidisciplinary, advanced technology
  - Often open source friendly

History of Migen



- Built a high-level language to describe programmable logic designs efficiently (2011)
- Originally only to satisfy the need for flexible metaprogramming of complex dataflows in graphics processors.
- Found out it was excellent for SoC, started MiSoC (2012)
- Now features
  - complete powerful language,
  - large library of cores,
  - own Pythonic simulation and co-simulation toolkit, and
  - support for dozens of platforms and toolchains.

Synchronous logic

```
a = Signal()
b = Signal()
x = Signal()
module.sync += x.eq(a | b)
verilog.convert(module)
```

Synchronous logic

```
module top(input sys_clk, input sys_rst);

reg a = 1'd0;
reg b = 1'd0;
reg x = 1'd0;

always @(posedge sys_clk) begin
  if (sys_rst) begin
    x <= 1'd0;
  end else begin
    x <= (a | b);
  end
end

endmodule
```

Finite state machines (FSMs)

```
fsm = FSM()
fsm.act("IDLE",
    foo.eq(a & b),
    If(start_munging, NextState("MUNGING"))
)
fsm.act("MUNGING",
    foo.eq(c),
    If(back, NextState("IDLE"))
)
```

Bus decoding/arbitration

```
cpu = LM32(...)
dma_engine = MungeAccelerator(...)
sdram = SDRAMController(...)
bus = BusCrossbar(
    # initiators
    [cpu.ibus, cpu.dbus, dma_engine.initiator],
    # targets
    [(0x10000000, sdram.bus),
     (0xc0000000, dma_engine.control)]
)
```

Again no magic - `BusCrossbar` is regular Python/FHDL

Implementation

```
from migen import *
from migen.build.platforms import m1

plat = m1.Platform()
led = plat.request("user_led")

m = Module()
counter = Signal(26)
m.comb += led.eq(counter[25])
m.sync += counter.eq(counter + 1)

plat.build(m)
```

Runs synthesis+PAR (ISE/Quartus/Lattice/Yosys, on Linux/Windows) and generates bitstream file.

MiSoC

- Provides high level classes for bus interconnect and MMIO:
  - Wishbone
  - CSR (as above)
  - streaming (ex-dataflow) interfaces
- Provides many cores:
  - Processors (wrapped Verilog): LM32, mor1kx (a better OpenRISC)
  - SDRAM controllers and PHYs (SDR, DDR1-3, fastest open source DDR3 controller @64Gbps)
  - UART, timer, SPI, 10/100/1000 Ethernet
  - VGA/DVI/HDMI framebuffer, DVI/HDMI sampler
- Provides bare-metal software (bootloader, low-level libraries) for your SoC.
- Provides SoC integration template classes.
- Provides basic and extensible SoC ports to FPGA boards.
- If those do not fit you, you can import the cores only and integrate yourself.

LTE base station

- PCIe x1 generic SDR board (Artix7 with AD9361: 70MHz to 6GHz)
- Almost 100% Migen/MiSoC code (the only exception is the PCIe transceiver wrapper)
- Designed to be coupled together for MIMO 4x4
- With software LTE stack: allows affordable LTE BaseStation (10x cheaper than traditionnal solutions)
- $>$ 50 boards already produced.

EnjoyDigital

LTE base station

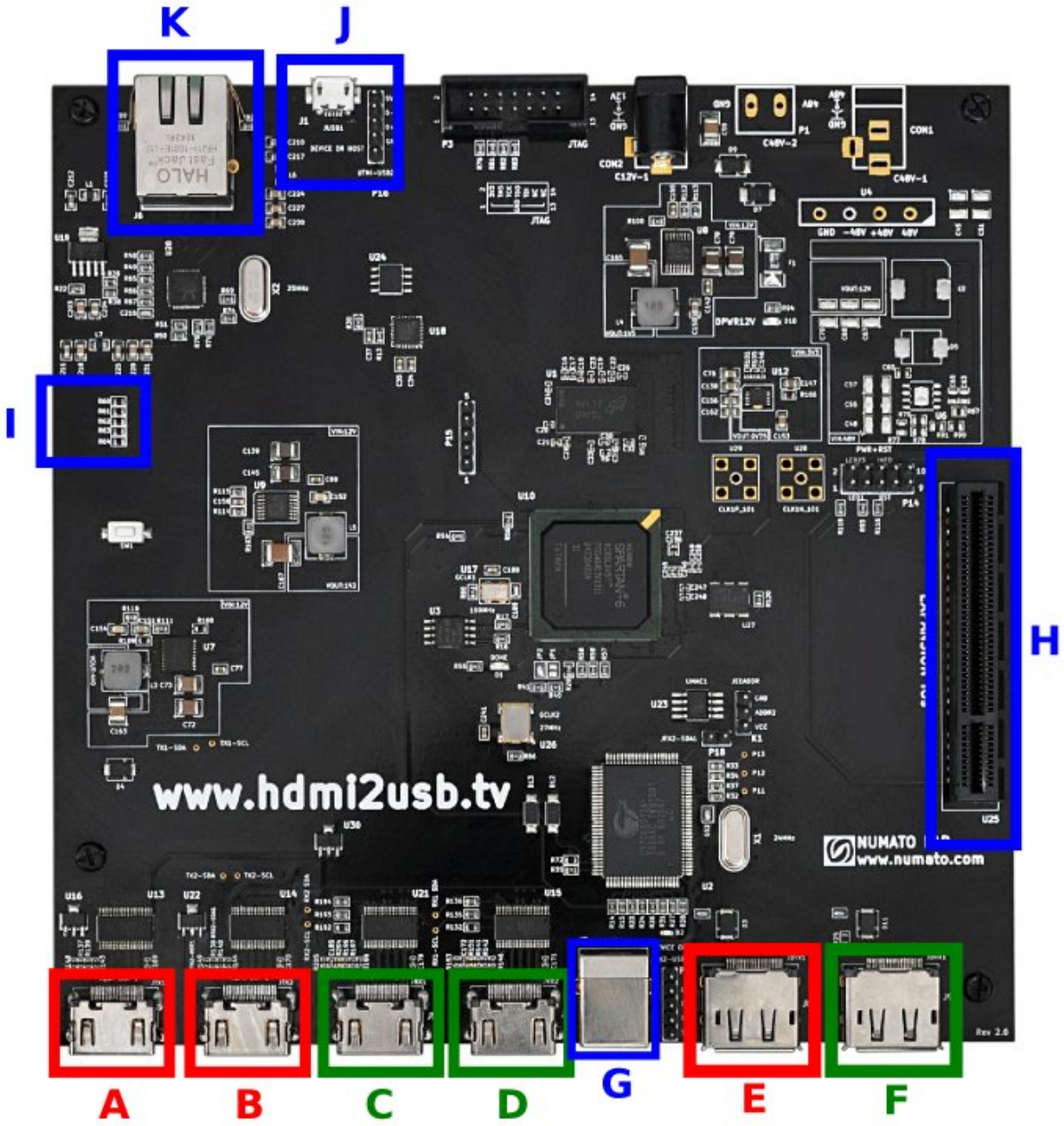
A few benefits of using Migen/MiSoC:

- Increased productivity compared with VHDL/Verilog.
- Developing a PCIe core would have been too expensive with traditional solutions, it has been done as part of this project.
- C header files that describes the hardware (registers/flags/interrupts) automatically generated.
- Kintex-7 KC705 prototyping board and Artix final board share most of the code.

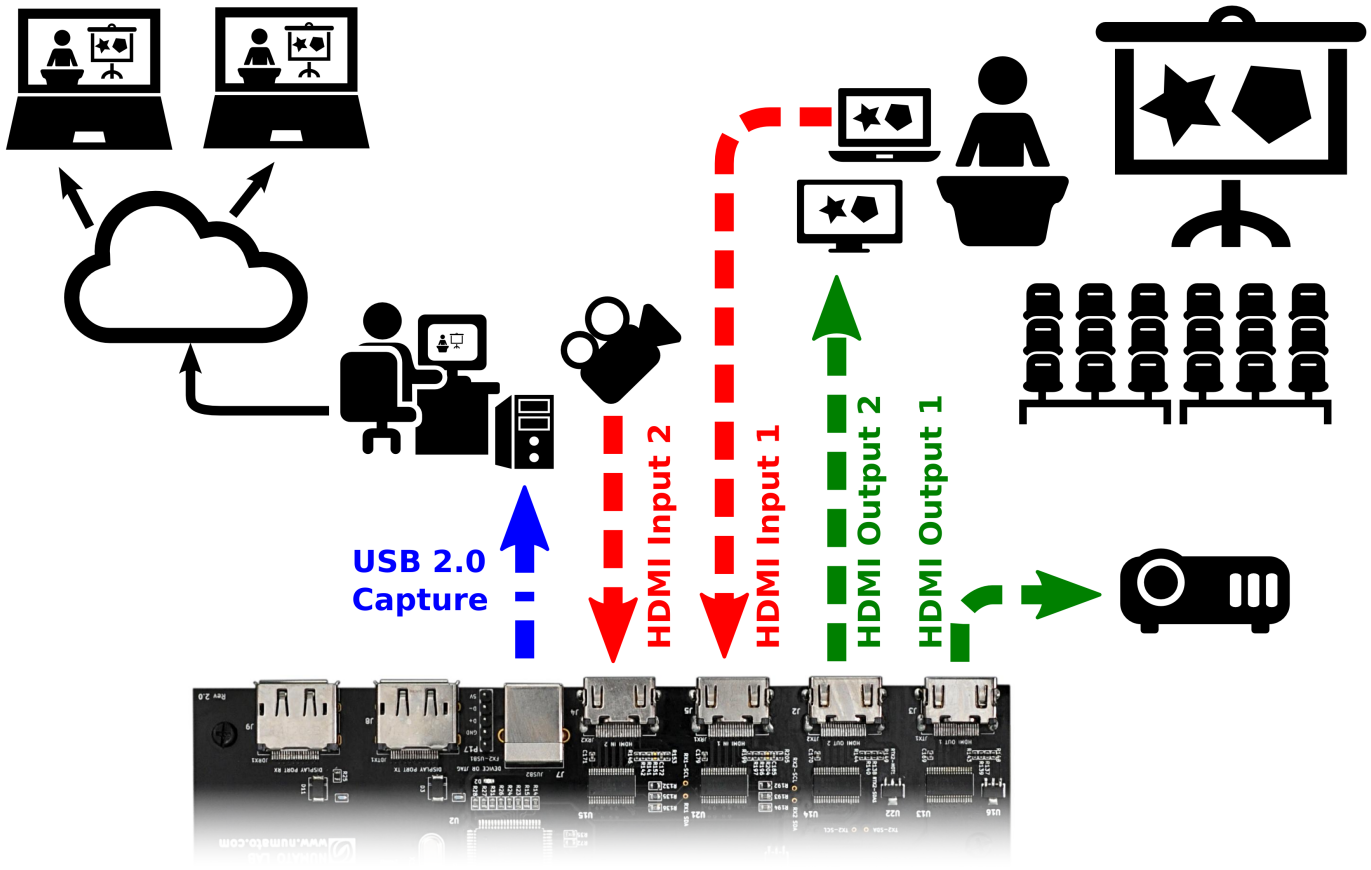
SATA 1.5/3/6G core

- Connect hard drives to FPGAs, 6Gbps per drive.
- Used in research project at University of Hong Kong.
- Kintex-7 FPGA (KC705).
- All Migen, including transceiver block instantiation.

Numato Opsis hardware

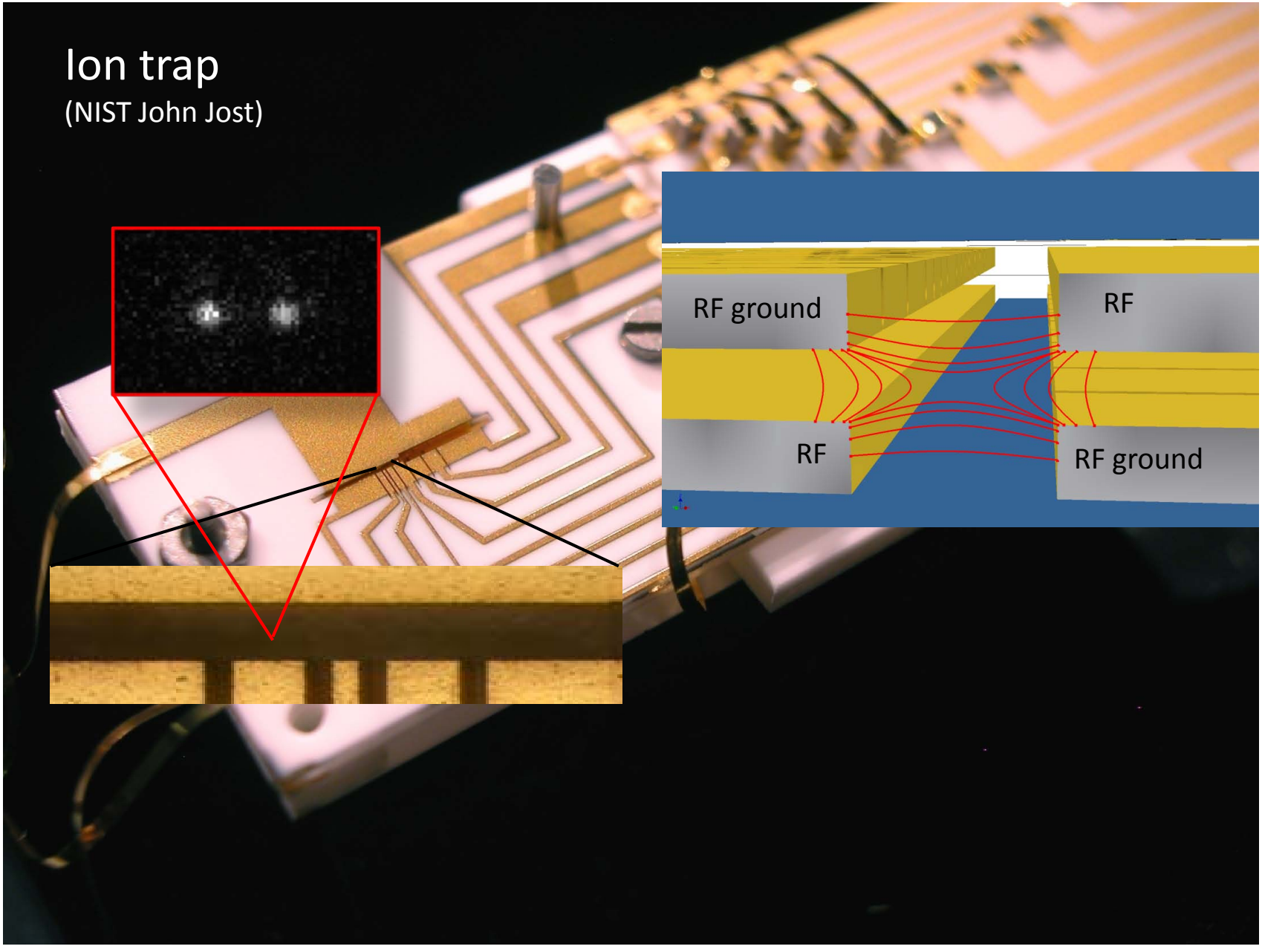


HDMI2USB project

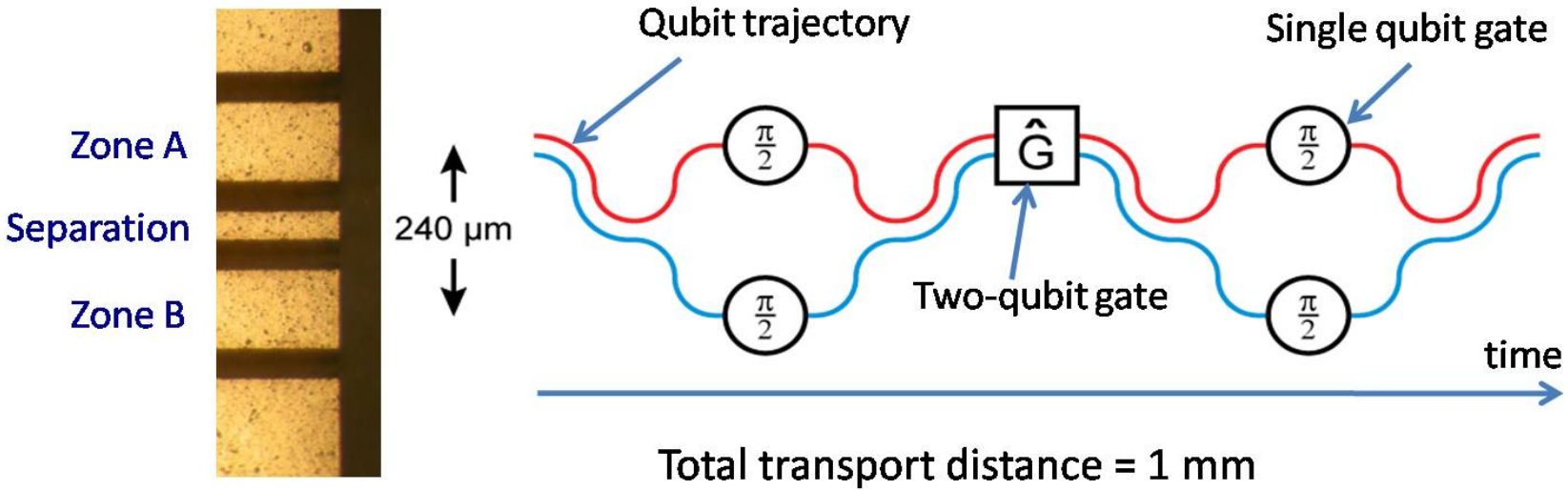


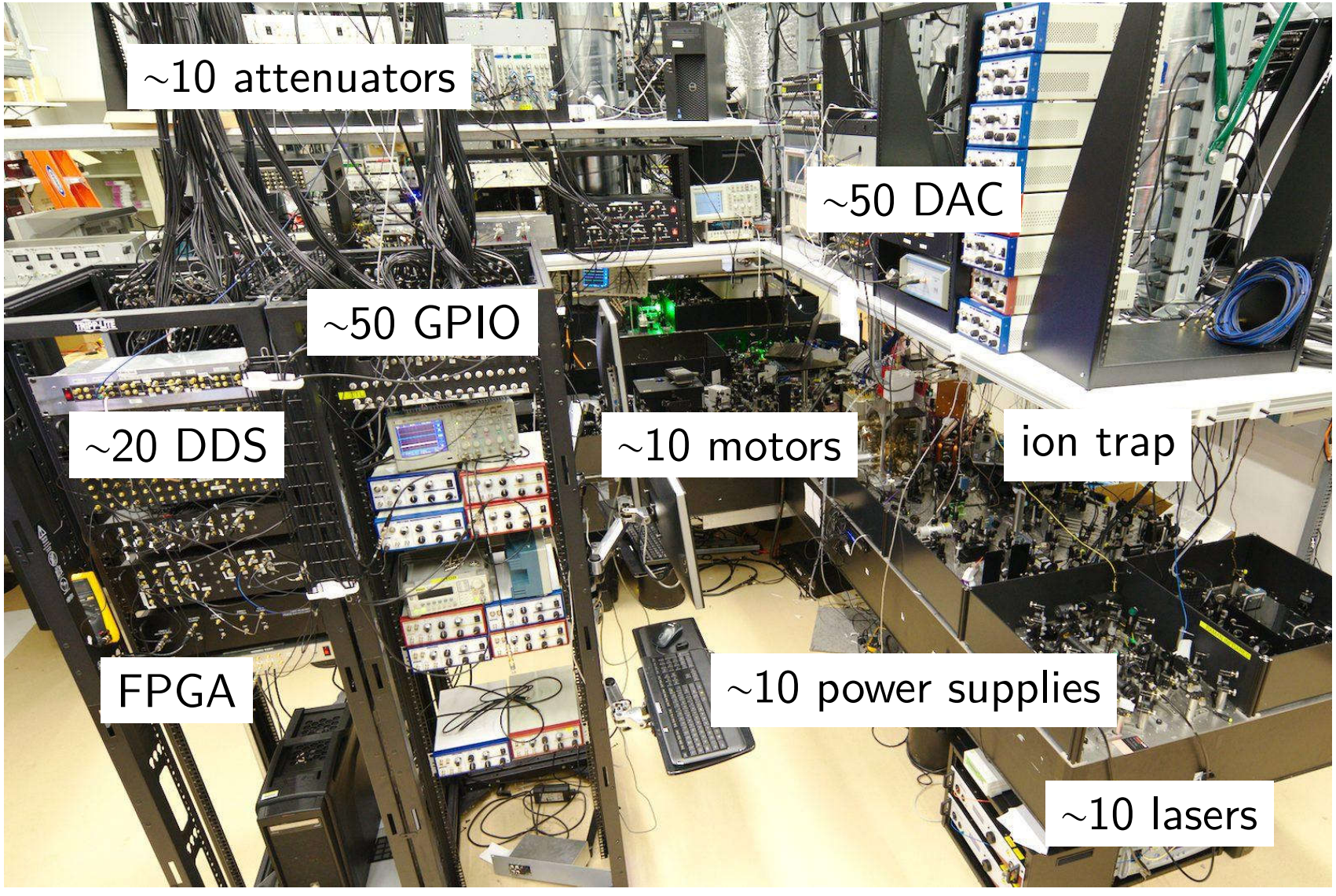
Ion trap

(NIST John Jost)



Quantum gate sequences (NIST)

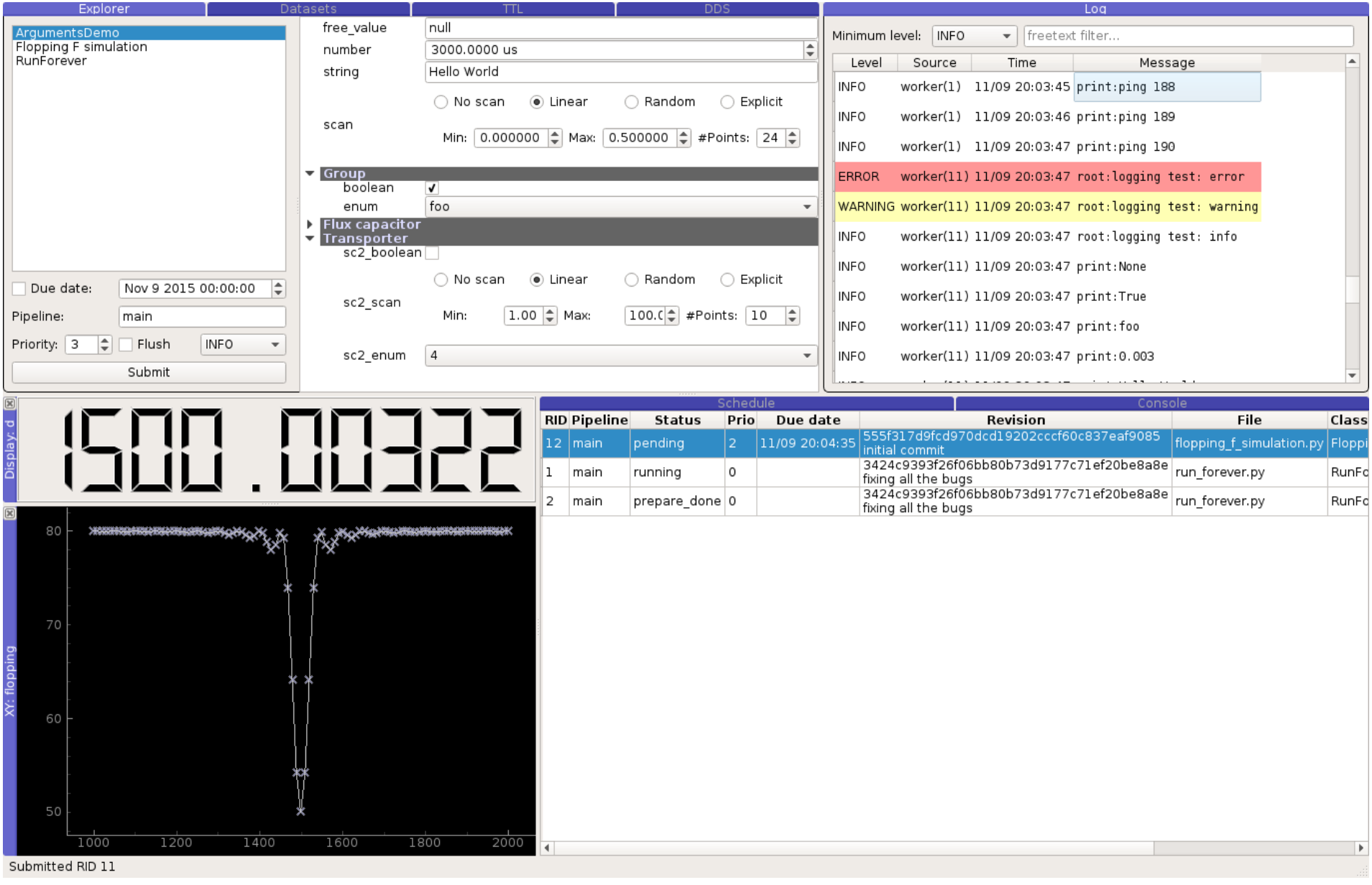



~10 attenuators
~50 DAC
~50 GPIO
~20 DDS
~10 motors
ion trap
FPGA
~10 power supplies
~10 lasers

ARTIQ

- ARTIQ is the **A**dvanced **R**eal-**T**ime **I**nfrastructure for **Q**uantum physics.
- An integrated software/gateware/hardware system that controls atomic physics experiments.
- Developed with the NIST Ion Storage Group (atomic clocks, quantum information, quantum simulation).
- Managing/scheduling experiments, driving distributed devices, analyzing/displaying/archiving results.
- Quickly and reliably deployable (Anaconda packages).

ARTIQ graphical user interface



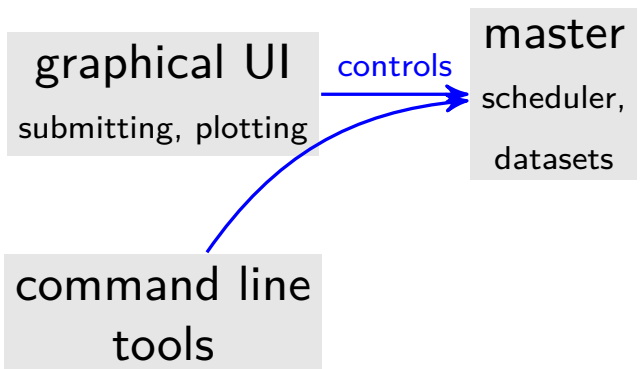
ARTIQ components

master

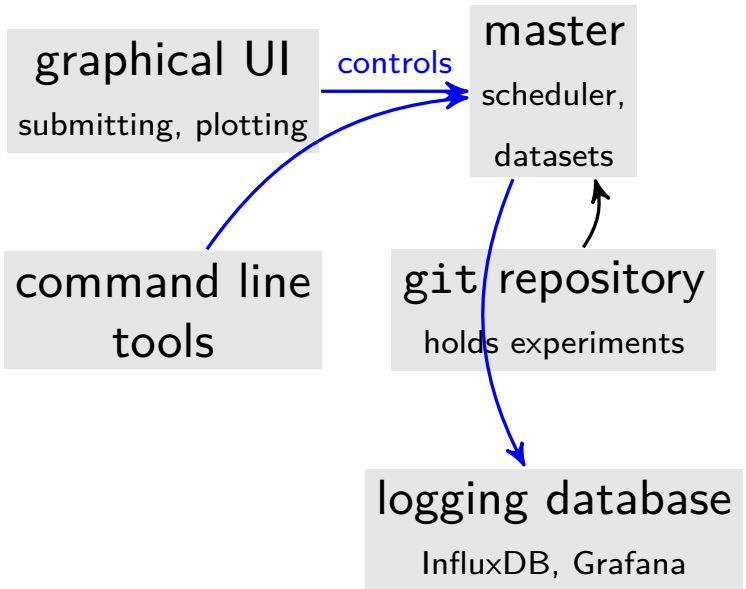
scheduler,

datasets

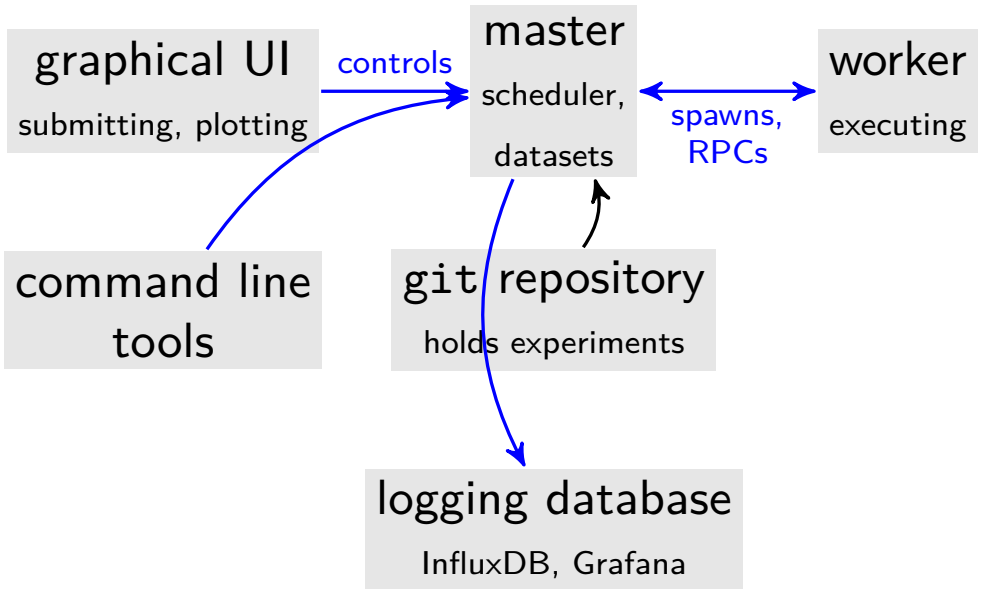
ARTIQ components



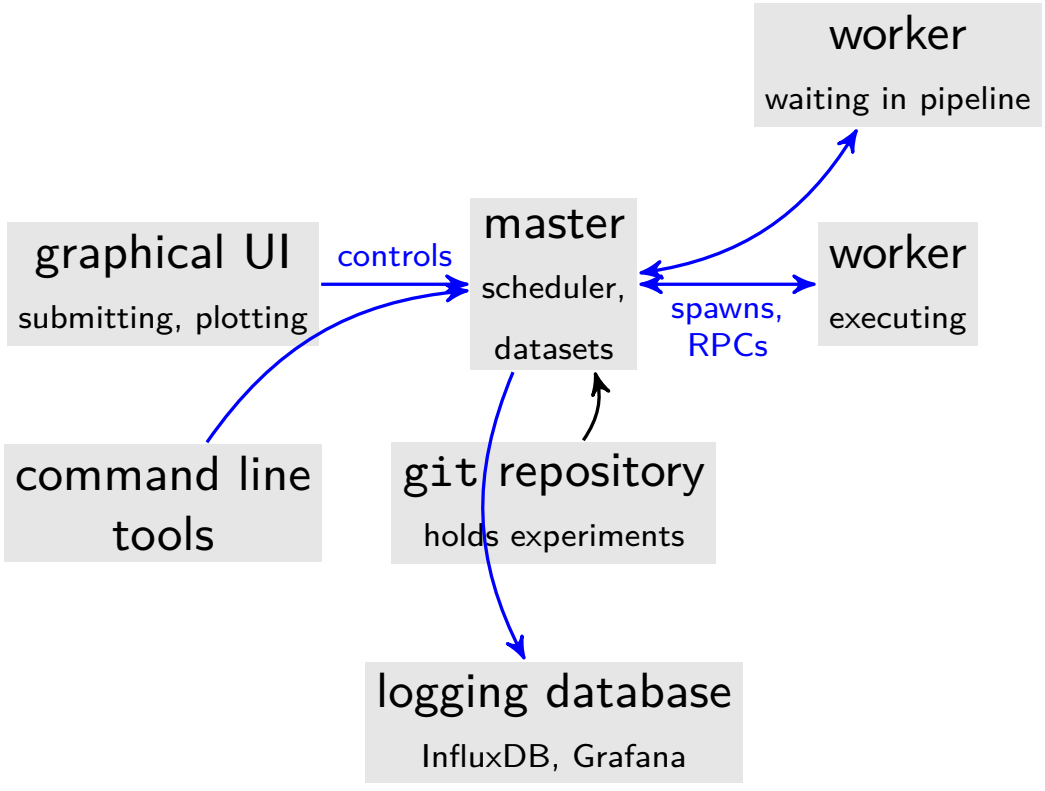
ARTIQ components



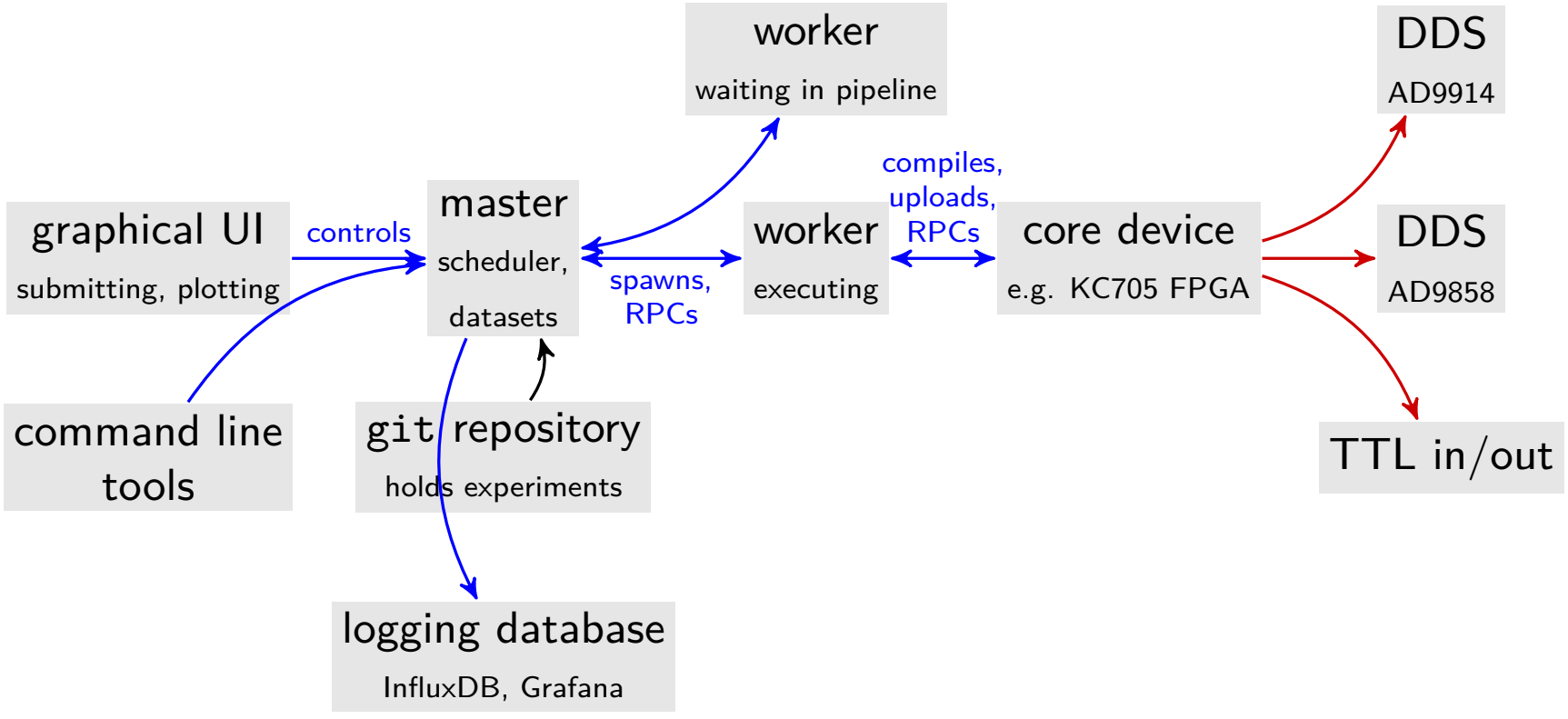
ARTIQ components



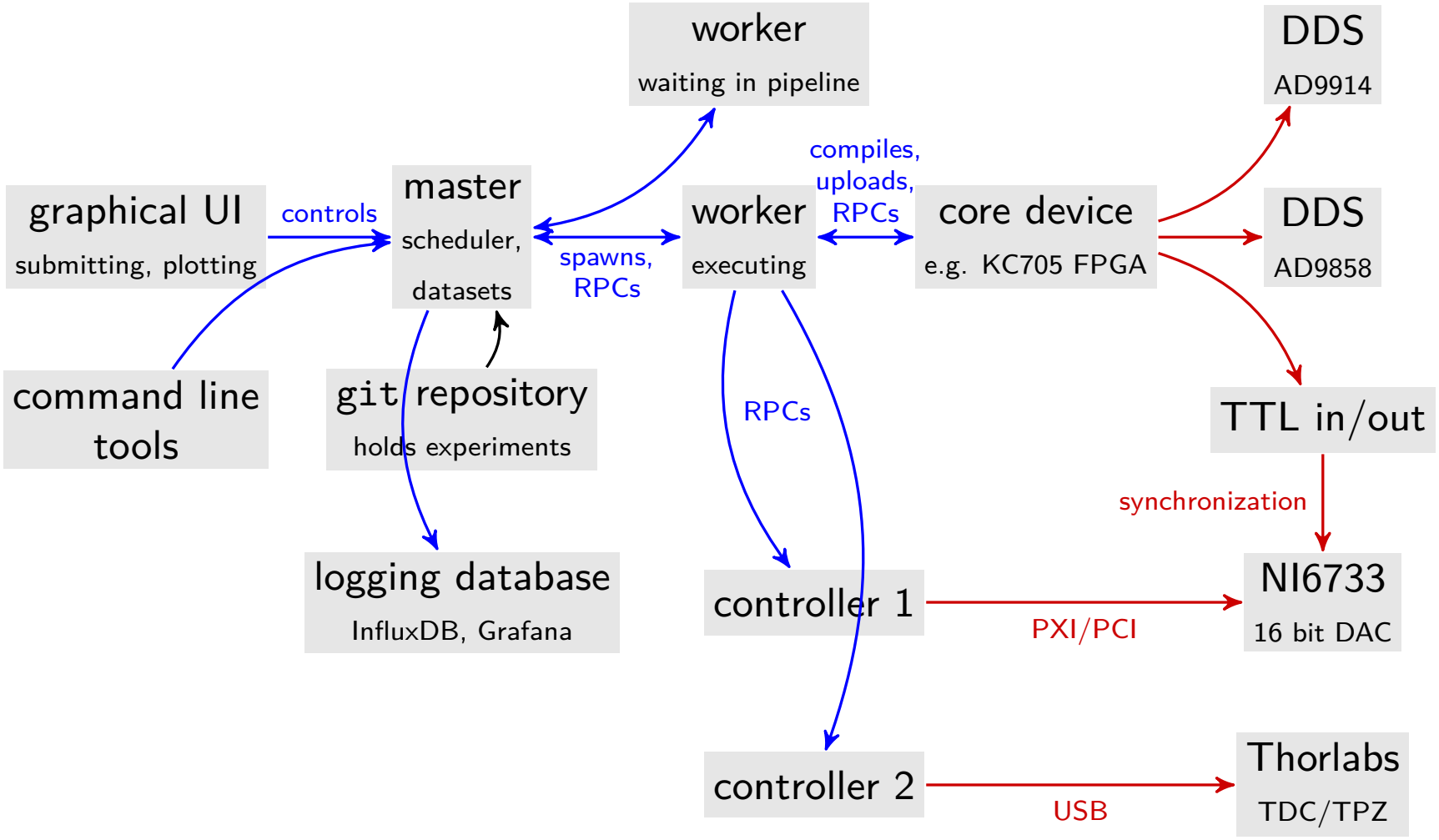
ARTIQ components



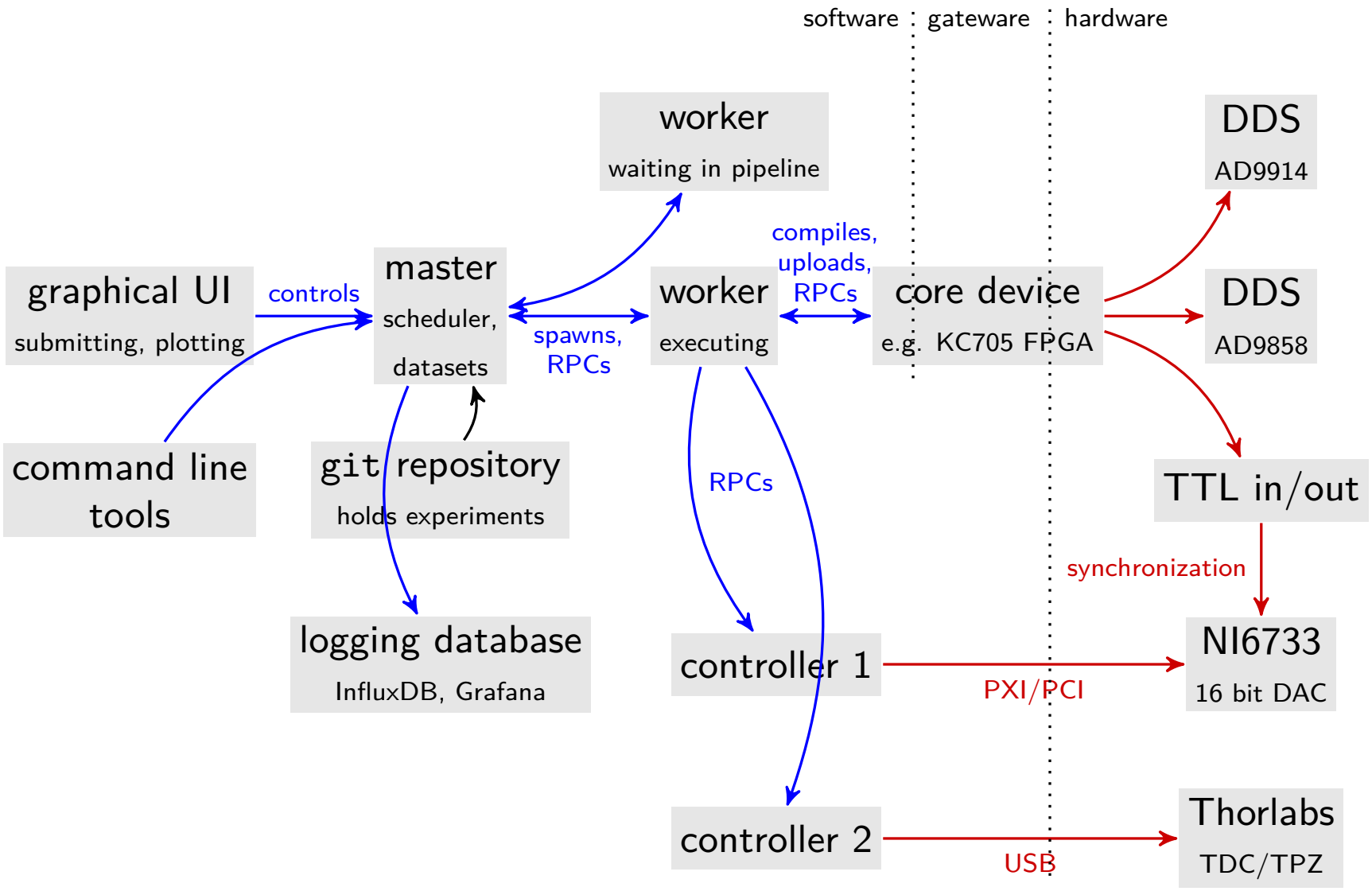
ARTIQ components



ARTIQ components



ARTIQ components



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

Physical quantities, hardware granularity

```
n = 1000
dt = 1.2345*ns
f = 345*MHz

dds.on(f, phase=0)                  # must round to integer tuning word
for i in range(n):
    delay(dt)                       # must round to native cycles

dt_raw = time_to_cycles(dt)         # integer number of cycles
f_raw = dds.frequency_to_ftw(f)     # integer frequency tuning word

# determine correct phase despite accumulation of rounding errors
phi = n*cycles_to_time(dt_raw)*dds.ftw_to_frequency(f_raw)
```

- Need well defined conversion and rounding of physical quantities (time, frequency, phase, etc.) to hardware granularity and back
- Complicated because of calibration, offsets, cable delays, non-linearities
- No generic way to do it automatically and correctly
- $\longrightarrow$ need to do it explicitly where it matters

Invite organizing experiment components and code reuse

```
class Experiment:
    def build(self):
        self.ion1 = Ion(...)
        self.ion2 = Ion(...)
        self.transporter = Transporter(...)

    @kernel
    def run(self):
        with parallel:
          self.ion1.cool(duration=10*us)
          self.ion2.cool(frequency=...)
        self.transporter.move(speed=...)
        delay(100*ms)
        self.ion1.detect(duration=...)
```

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

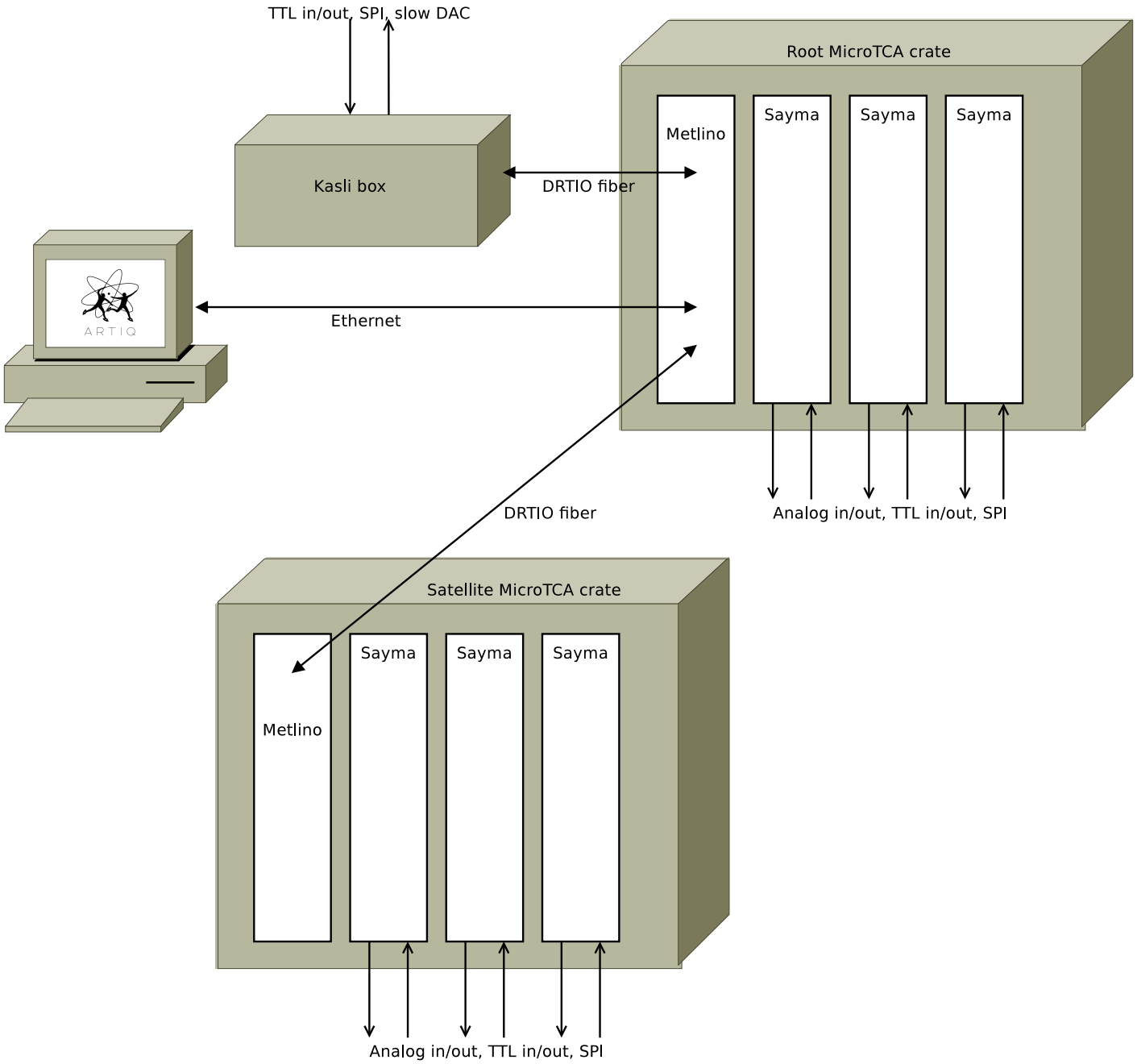
Kernel deployment to the core device

- RPC and exception mappings are generated
- Constants and small kernels are inlined
- Small loops are unrolled
- Statements in parallel blocks are interleaved
- Time is converted to RTIO clock cycles
- The Python AST is converted to LLVM IR
- The LLVM IR is compiled to OpenRISC machine code
- The OpenRISC binary is sent to the core device
- The runtime in the core device links and runs the kernel
- The kernel calls the runtime for communication (RPC) and interfacing with core device peripherals (RTIO, DDS)

Higher level features

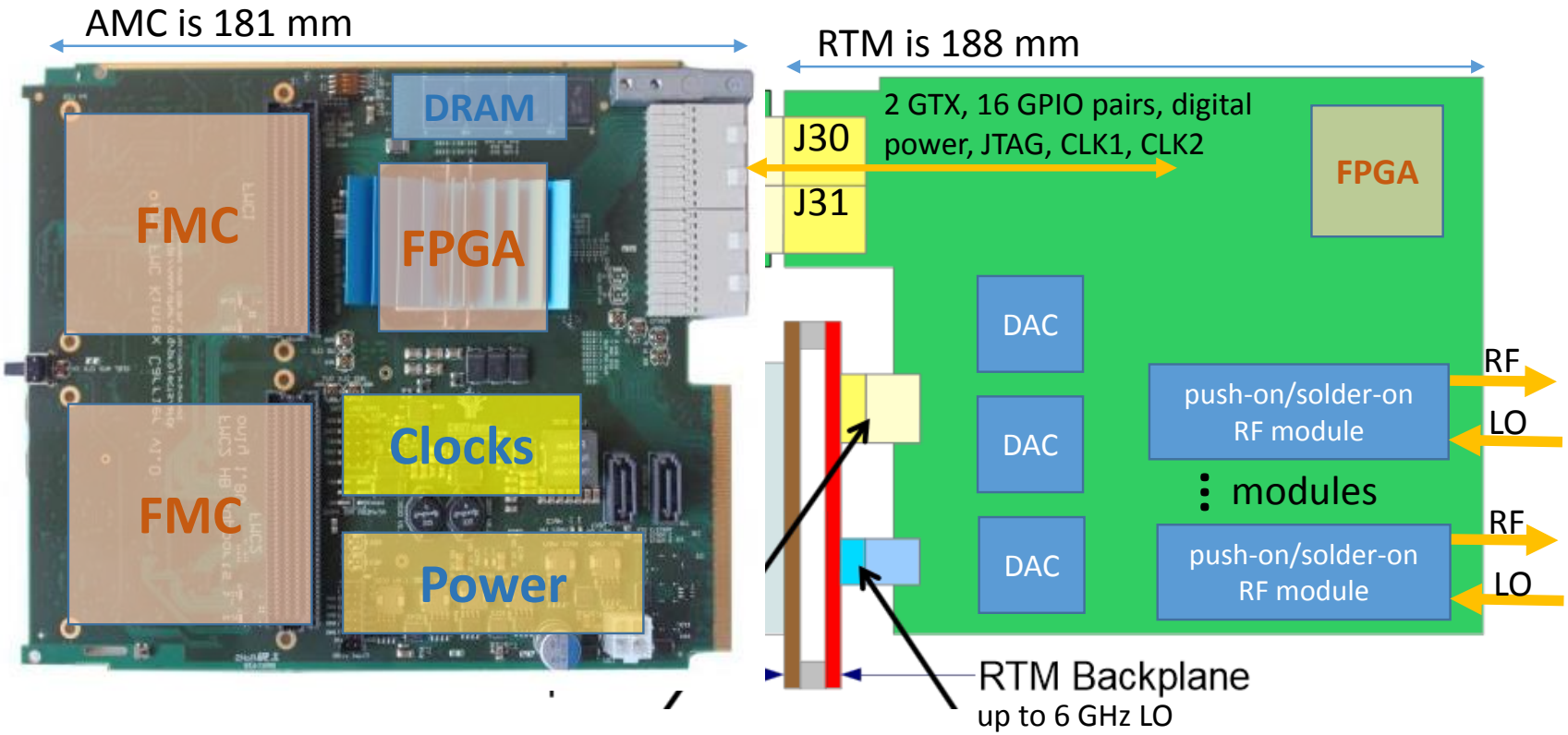
- Device management: drivers, remote devices, device database
- Parameter database
  e.g. ion properties such as qubit flopping frequency
- Scheduling of experiments
  e.g. calibrations, queue
- Archival of results (HDF5 format)
- Graphical user interface
  run with arguments, schedule, real-time plotting

SINARA


TTL in/out, SPI, slow DAC
Root MicroTCA crate
Metlino
Sayma
Sayma
Sayma
Kasli box
DRTIO fiber
ARTIQ
Ethernet
DRTIO fiber
Analog in/out, TTL in/out, SPI
Satellite MicroTCA crate
Metlino
Sayma
Sayma
Sayma
Analog in/out, TTL in/out, SPI

Agile RF for ARTIQ v5

JESD204B, 2.5 GSPS



TI DAC38J84

- 16-bit resolution
- 2.5 GSPS output (input 1.25 GSPS)
- 12.5 GBPS, JESD204B, 8-lane
  - needs 7-lanes/chip
- ¾-wire SPI
- 144 FC BGA package (10.1 mm square)
- 1.8 W power dissipation
- 14*4 + 2 fast IO, 13 other IO = 71 IO

DAC Resources

- 3 of DAC38J84 ($89/ea)
- clock distribution at 2.5 GHz
- 21 GTX transceivers
- 6 W dissipation at 3.3V
- also 0.9V, 1.8V for digital

FPGA

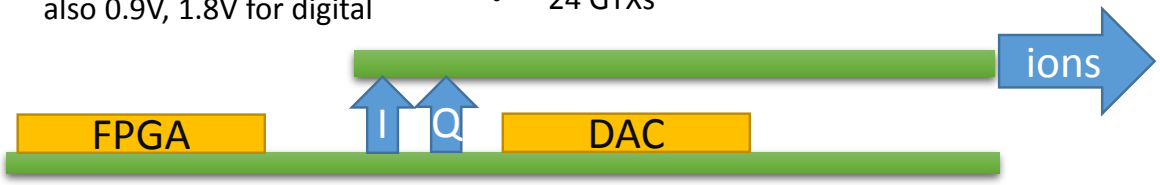
- XC7K355T $2.5k/ea
- 356k LUT
- 300 User IO
- 24 GTXs

Logic to J30/J31

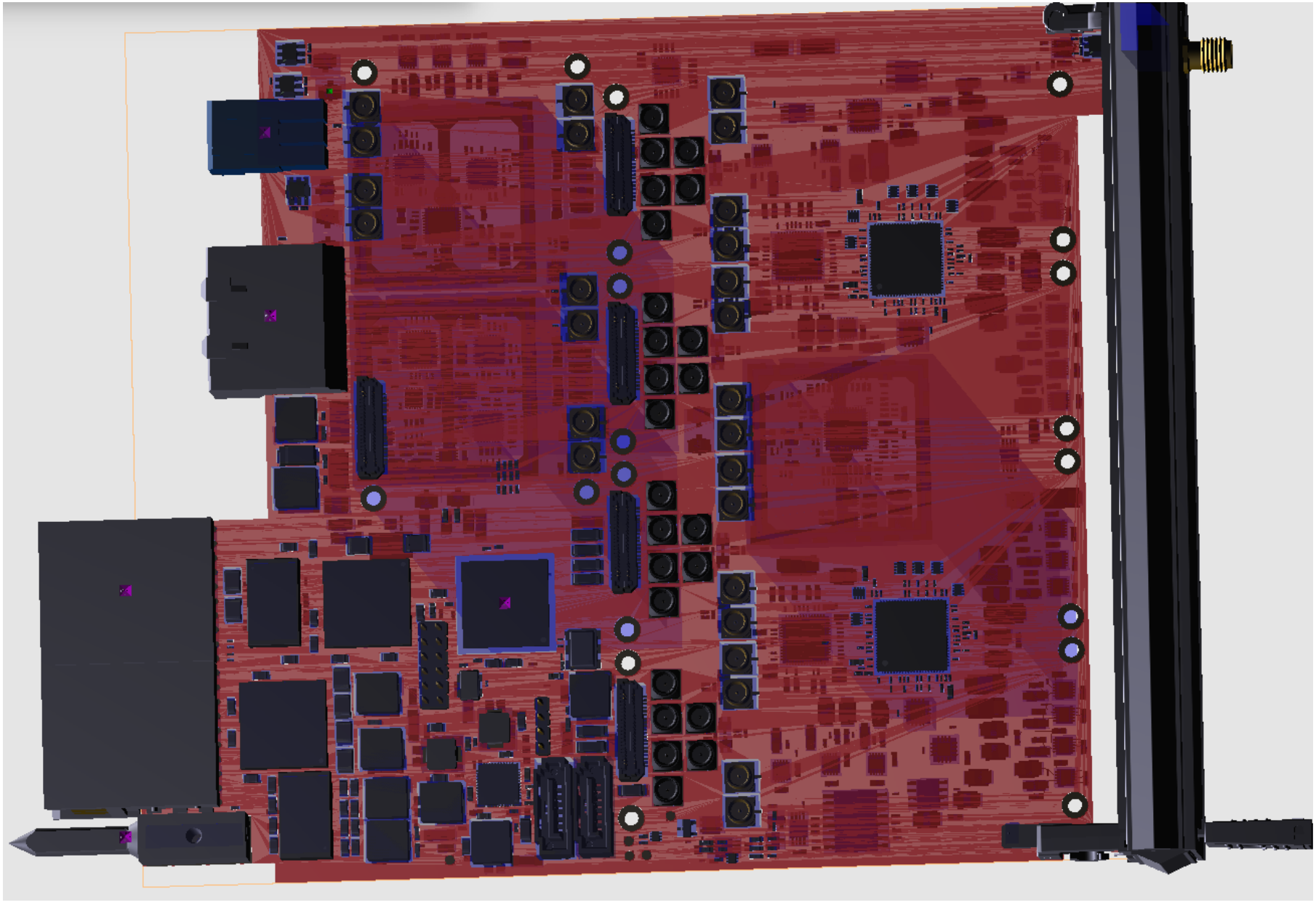
- 2 GTX
- 16 GPIO

Other Logic

1 GTX to SFP+

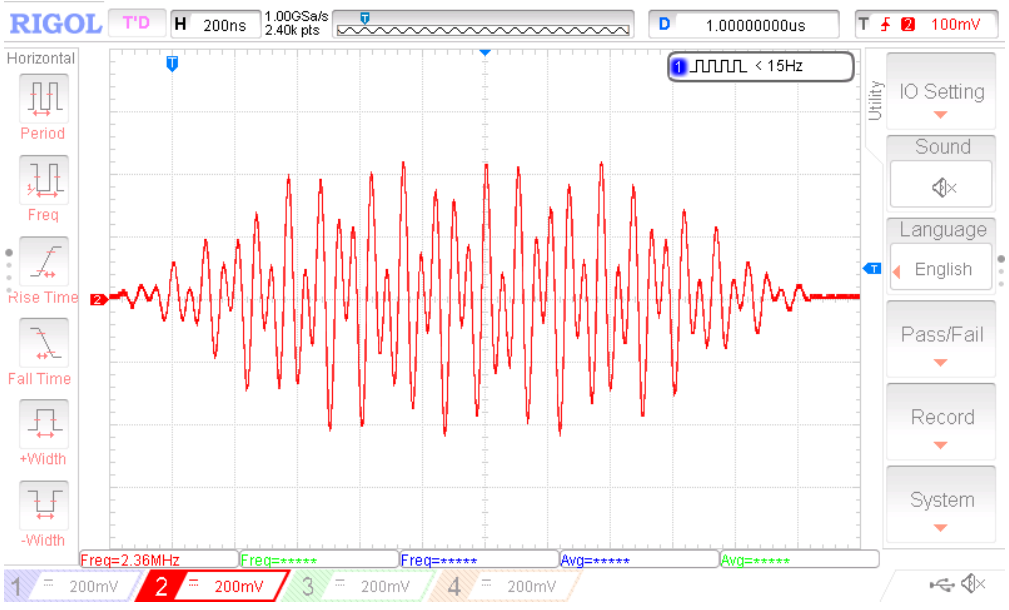


Sayma RTM



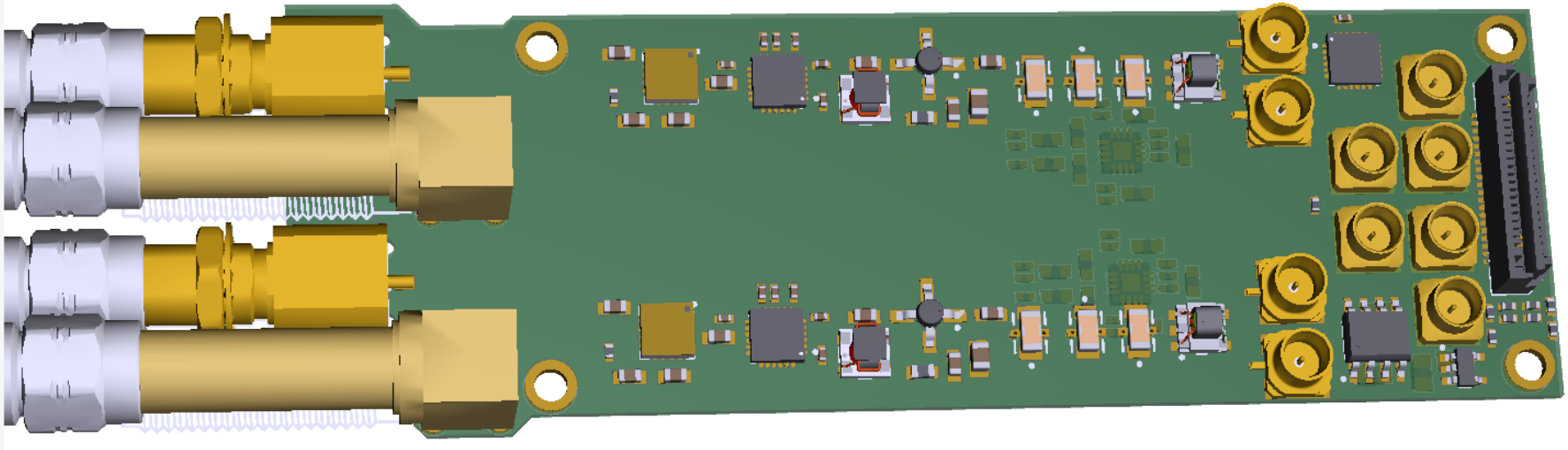
PHASER

- Smart Arbitrary Waveform Generator (SAWG) gateware for SAYMA
- GHz sample rate, interpolating up to 2.4 GS/s
- Multiple Digitally Controlled Oscillators per channel
- Interpolating in phase, amplitude, and frequency using B-Splines



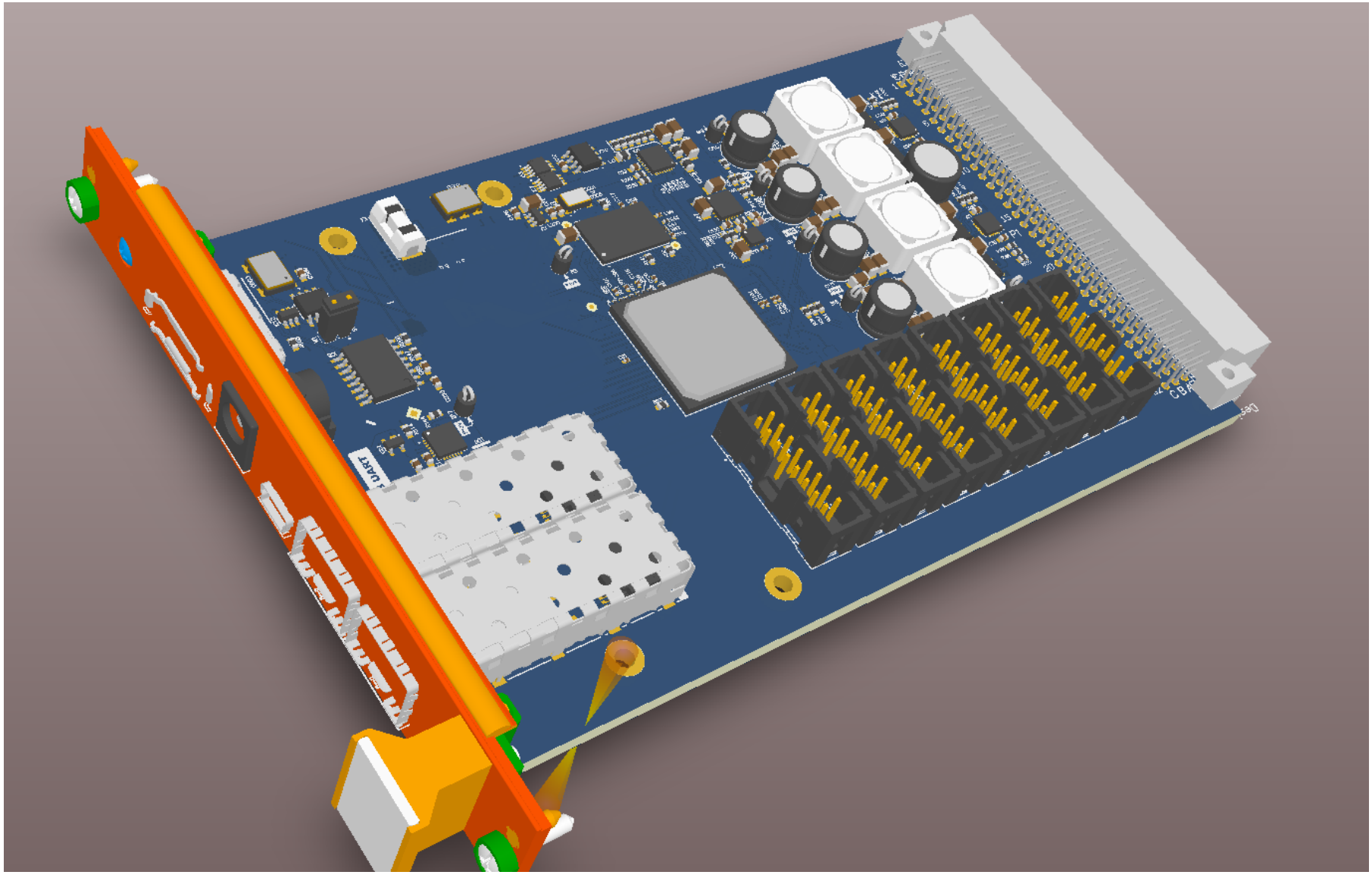
Sayma Analog Frontend Mezzanine

Modular analog mezzanines (application dependent): DC signals, filtering, IQ upconversion/downconversion



Kasli

Low-cost distributed I/O satellite



Kasli Crate

SPI/I2C/TTL/CameraLink input/output options