Warsaw University of Technology

Experiment Control with the ARTIQ/Sinara Control Ecosystem

Paweł Kulik
Mikołaj Sowiński
Jakub Matyas

Quantum Sensing autumn school 2024 - CERN, Geneva

About us

The Faculty of Electronics and Information Technology

Warsaw University of Technology

- **Electronics for High Energy and Quantum Physics research group**

  https://github.com/elhep

- **Sinara Open-Hardware Project**

  https://github.com/sinara-hw/

Disclaimer

ARTIQ is originally authored by *Bourdeauducq, Sébastien et al. (2016). ARTIQ 1.0. Zenodo. 10.5281/zenodo.51303* and currently is maintained and developed by M-Labs with and the community.

Organization

Slides and source codes

https://github.com/elhep/artiq_dax_tutorial_materials

ARTIQ 8 manual (used for this tutorial):

https://m-labs.hk/artiq/manual/introduction.html

Glossary

- ARTIQ - Advanced Real-Time Infrastructure for Quantum physics

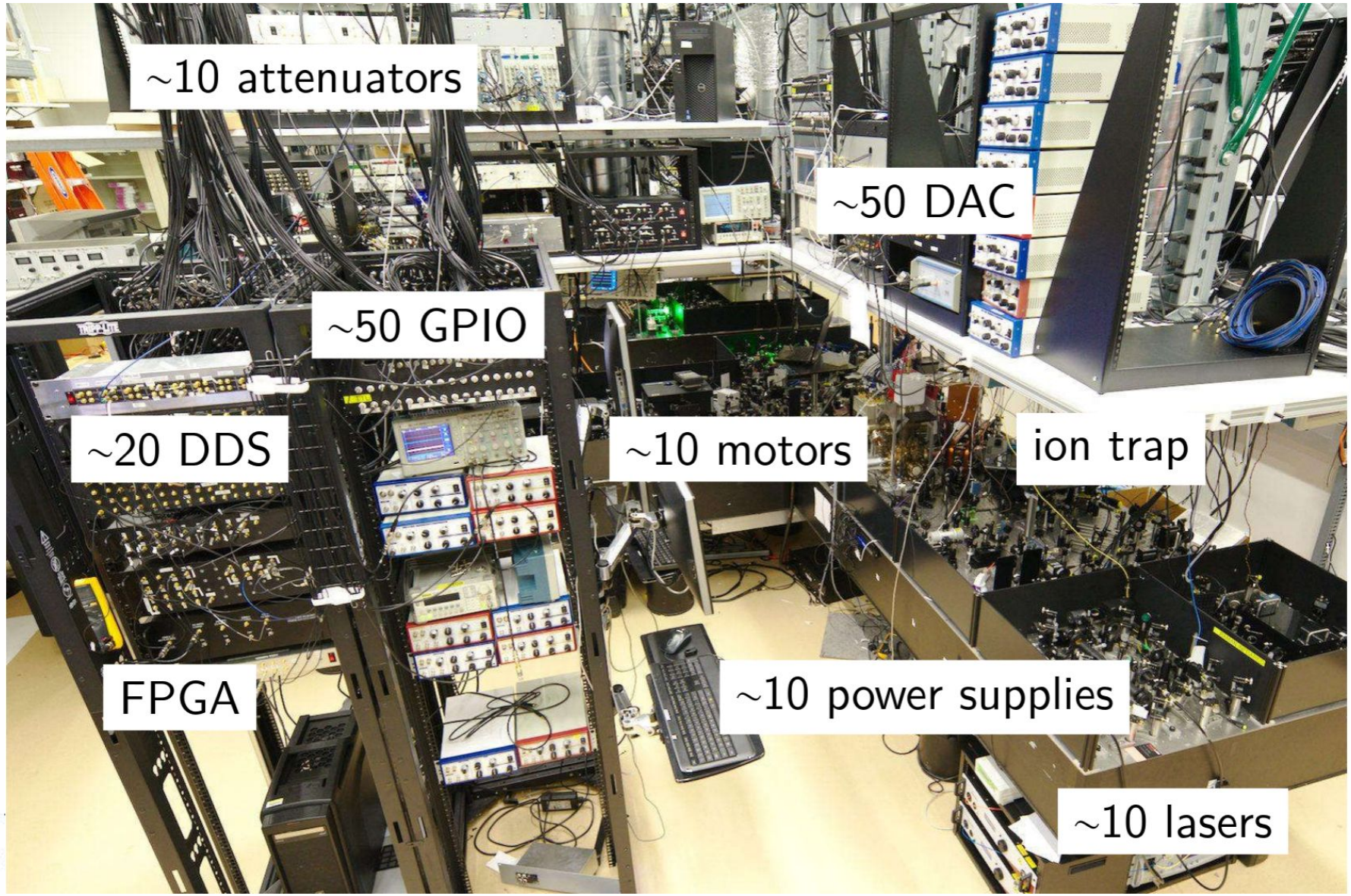
ARTIQ

Motivation for ARTIQ

- Experiments have a lot of requirements that are hard to get right on your own
  - Precise timing
  - Low latency
  - Flexibility
  - Extensibility
- No vendor lock-in
- Deduplicating efforts between labs
- User agency - change underlying project, extend, contribute

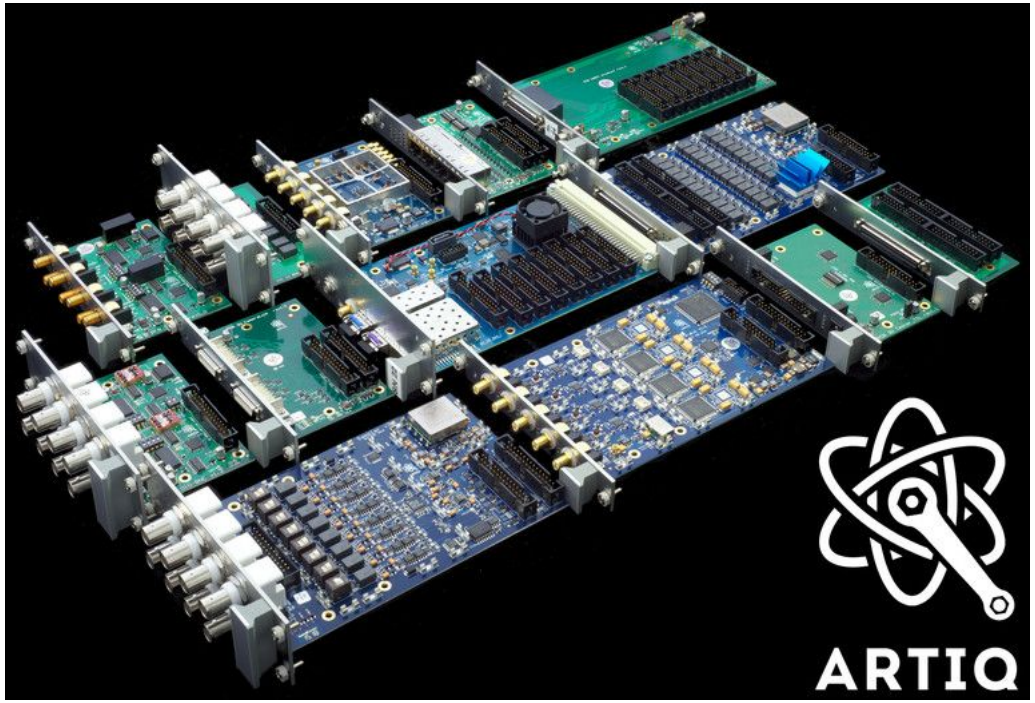
ARTIQ

Motivation for ARTIQ



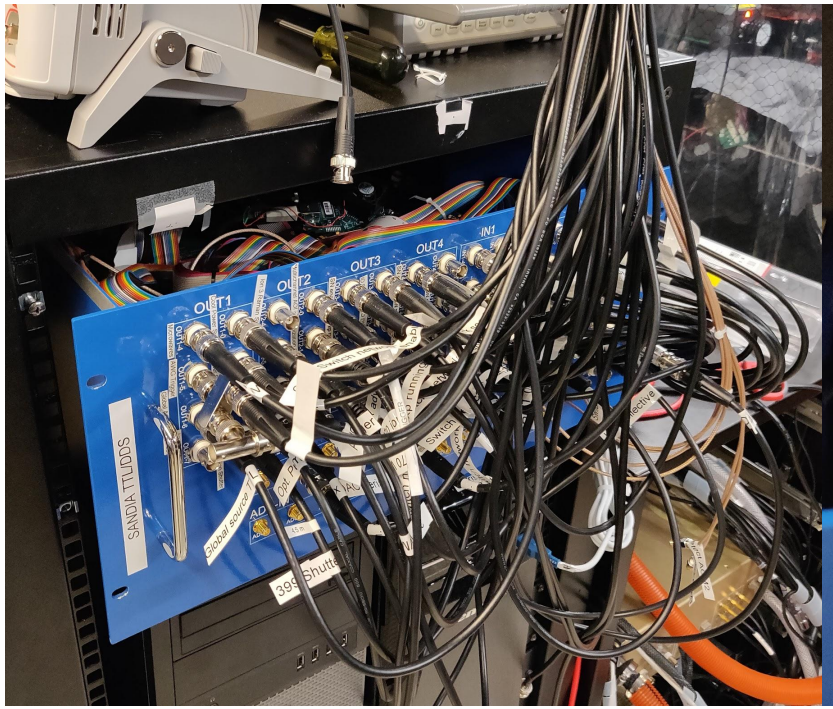
Glossary

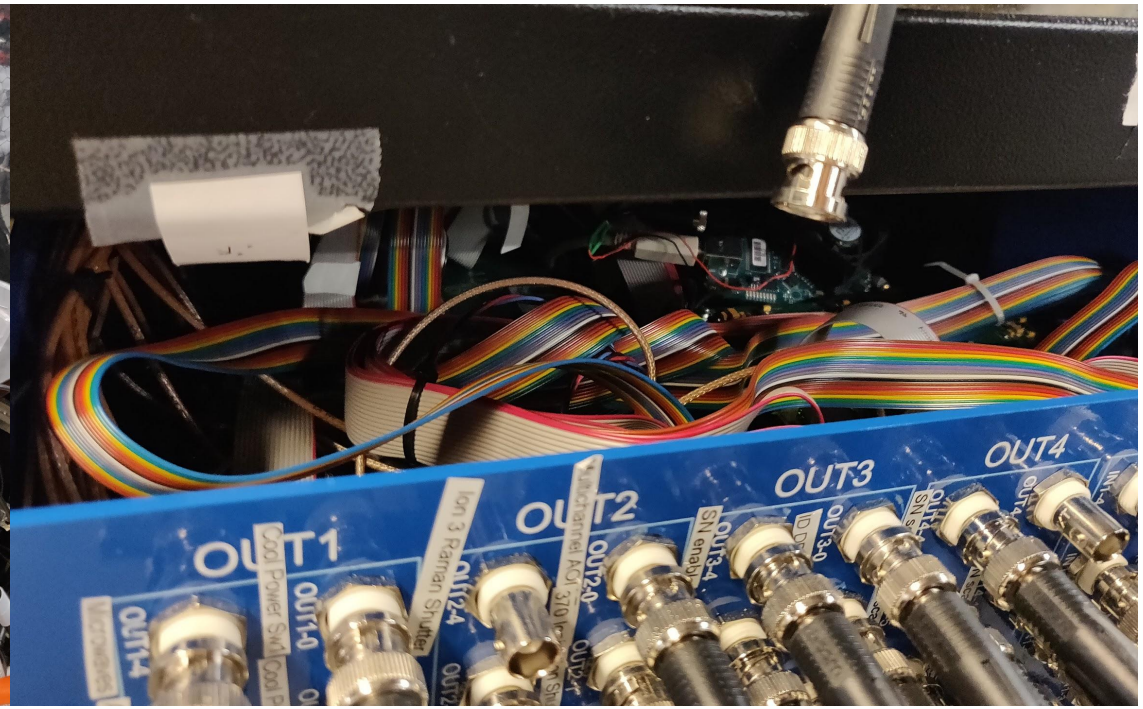
- ARTIQ
- Sinara - Hardware designed for ARTIQ



Motivation for Sinara

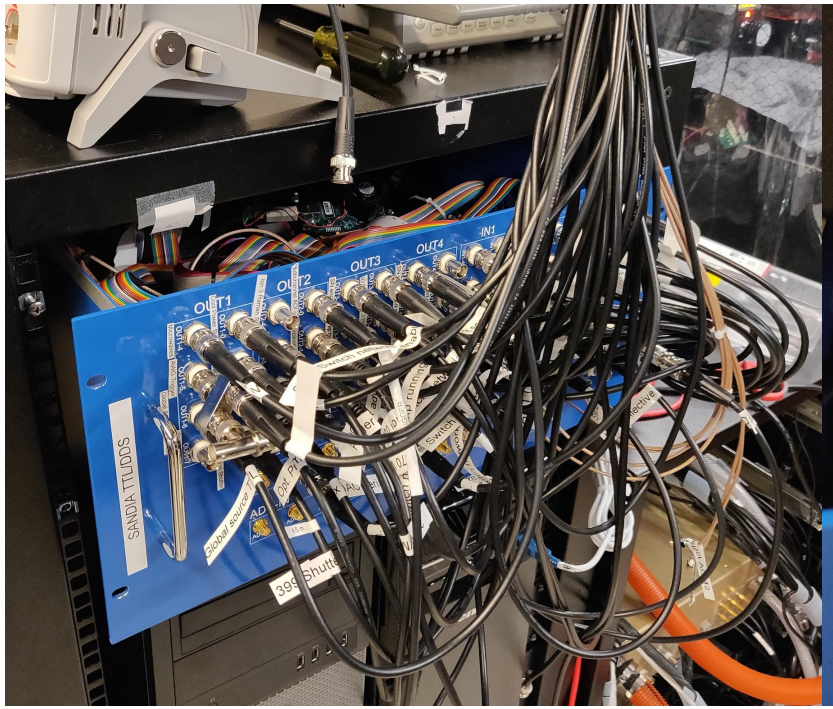
- Physicists are not engineers - and that's OK!
- If you're not careful, you'll end up with this:

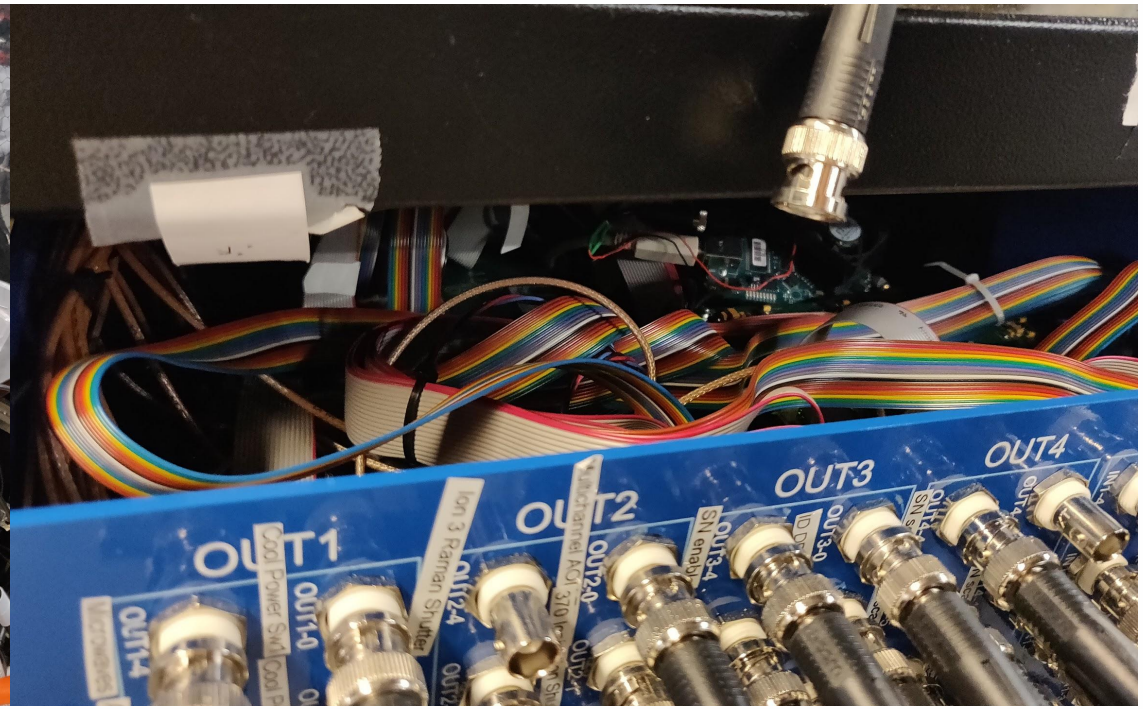




Motivation for Sinara

- Physicists are not engineers - and that's OK!
- If you're not careful, you'll end up with this:
- Worst case scenario - this was done years ago by someone who graduated



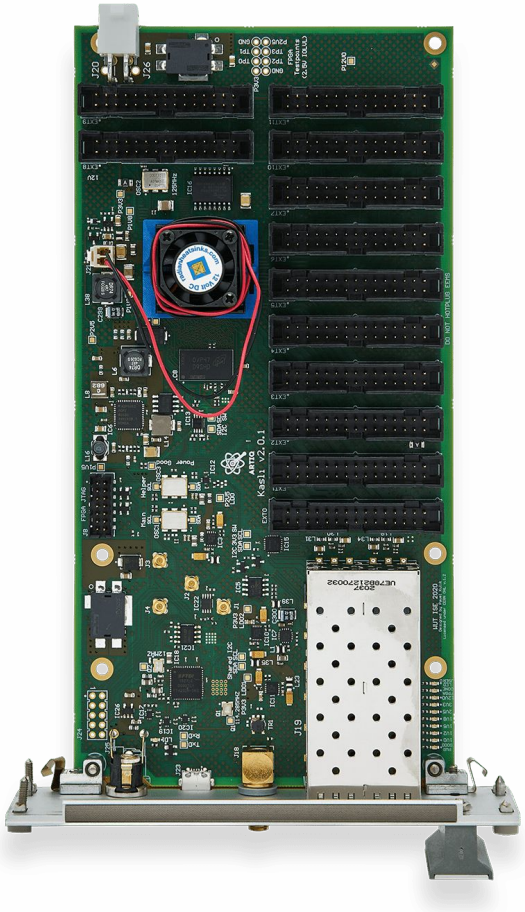


Motivation for Sinara

- Again - no vendor lock-in
- Deduplicating efforts between labs
- User agency - change underlying project, extend, contribute
- Dream of enabling experiment reproducibility across laboratories

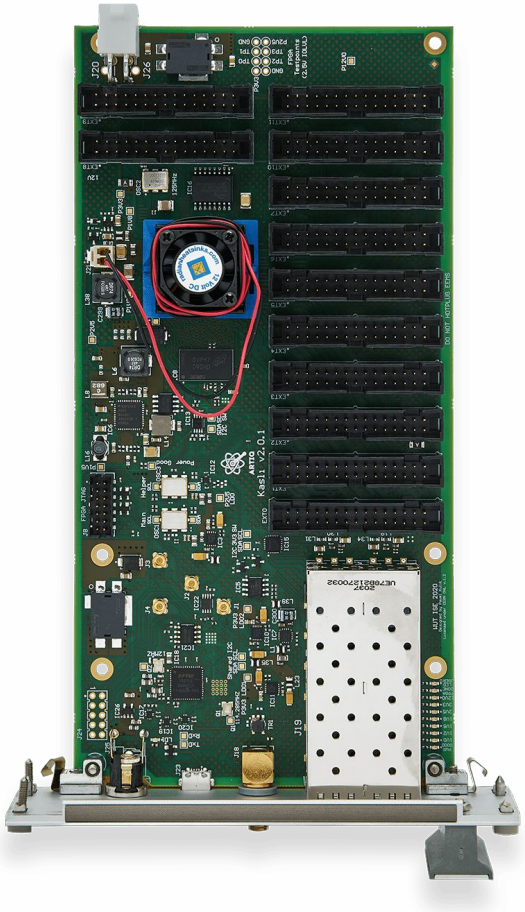
Glossary

- ARTIQ
- Sinara
- Controller



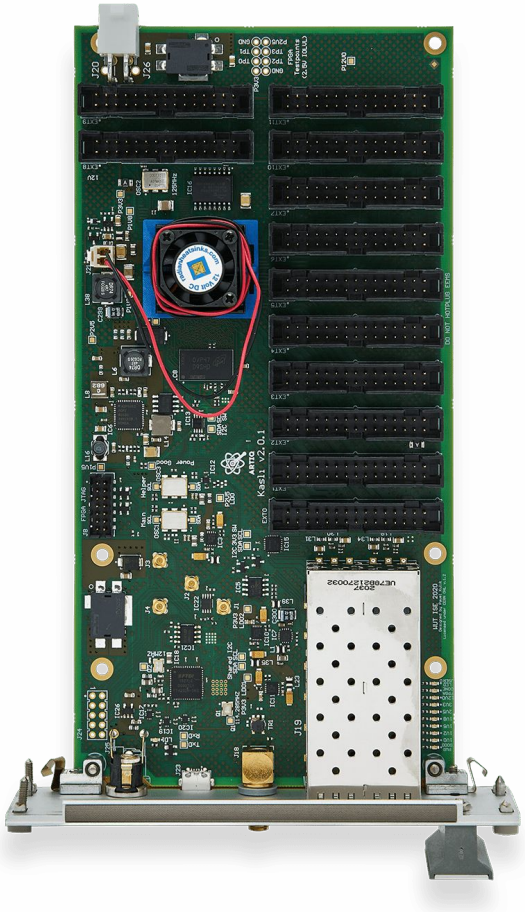
Glossary

- ARTIQ
- Sinara
- Controller
- Satellite - secondary controller



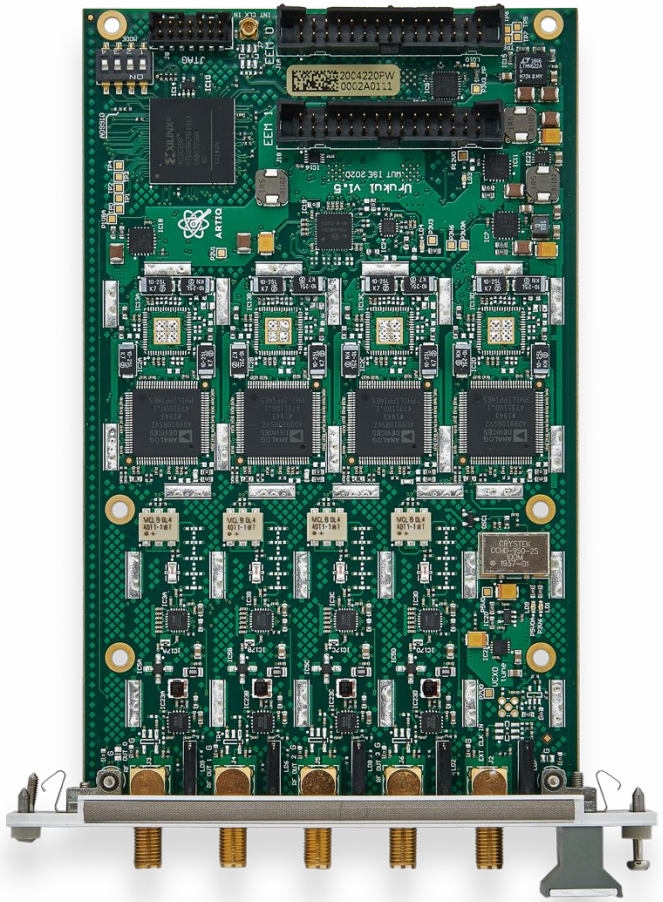
Glossary

- ARTIQ
- Sinara
- Controller
- Satellite
- Gateware - firmware for the FPGA



Glossary

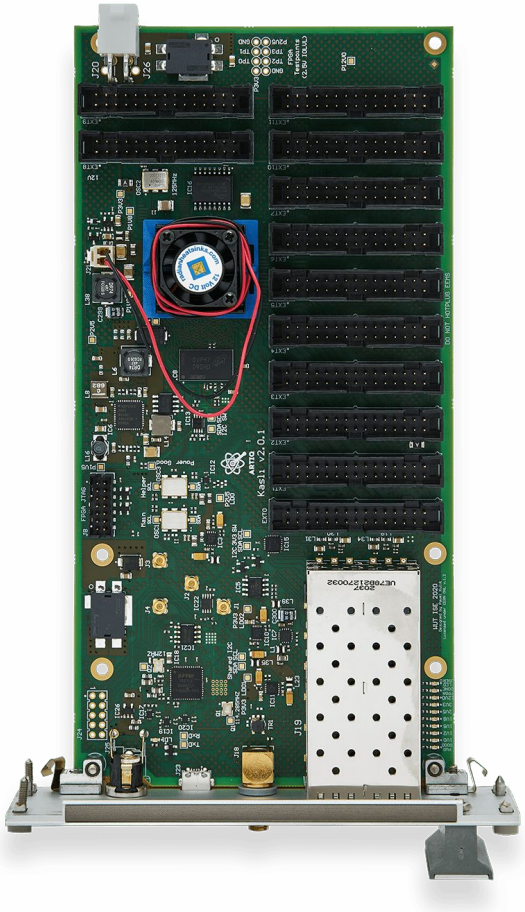
- ARTIQ
- Sinara
- Controller
- Satellite
- Gateware
- EEM



Warsaw University of Technology

Glossary

- ARTIQ
- Sinara
- Controller
- Satellite
- Gateware
- EEM
- hard real-time - all or nothing



Warsaw University of Technology

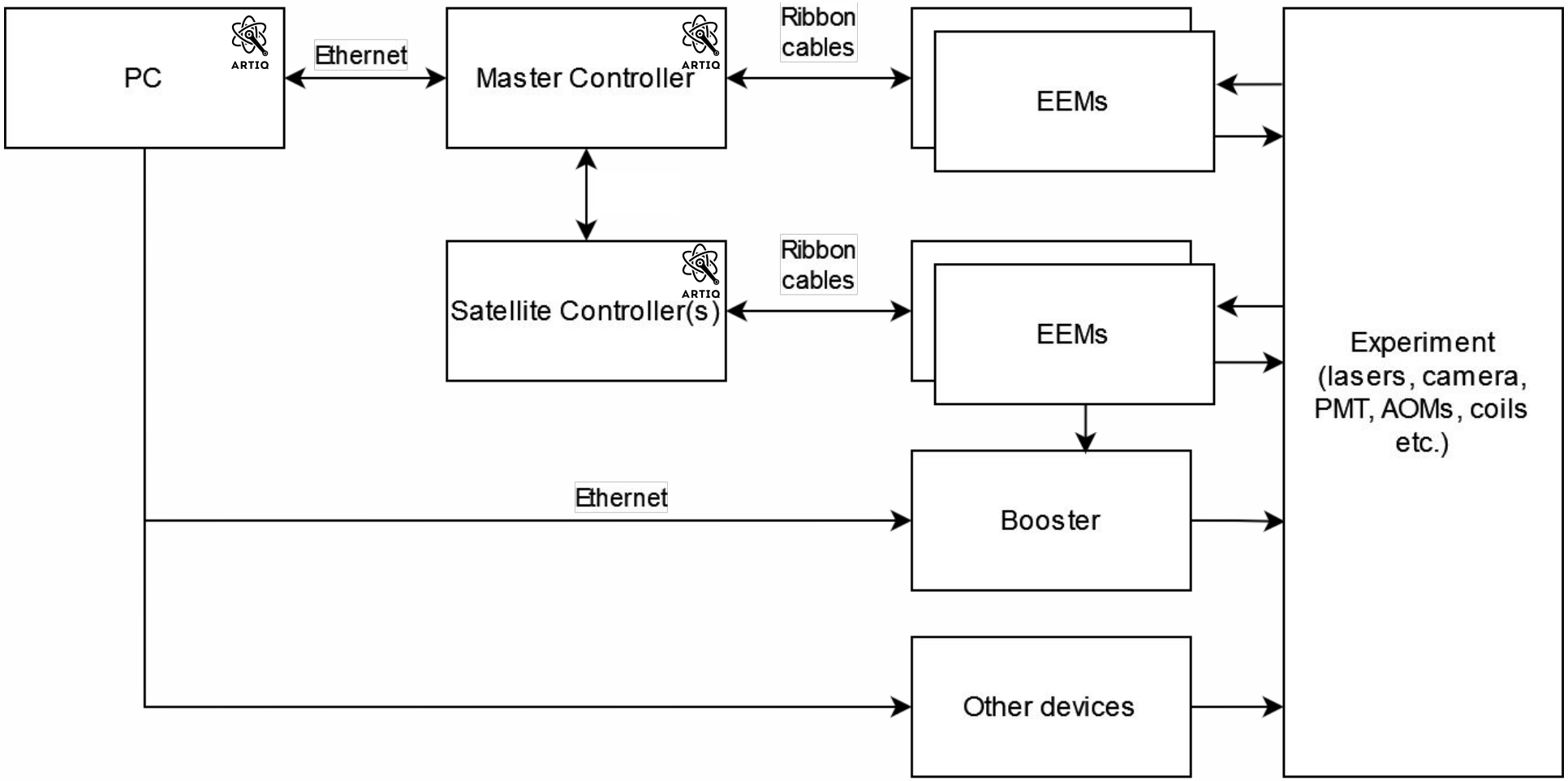
What does ARTIQ code look like?

- Python-based DSL
- Stages:
  - **Build** - devices used, parameters, **no hardware access**
  - **Prepare** - pre calculating values, **no hardware access**
  - Run - perform operations **on hardware**
  - **Analyze** - analyze data gathered during run stage, **no hardware access**

```
from artiq.experiment import *


class Experiment(EnvExperiment):

    def build(self):
        self.setattr_device("core")
        self.setattr_device("ttl1")

    def prepare(self):
        self.foo = [i%2 for i in range(100)]

    @kernel
    def run(self):
        self.core.reset()
        self.ttl1.pulse(100*ns)

    def analyze(self):
        result = sum(self.foo)
```

Warsaw University of Technology

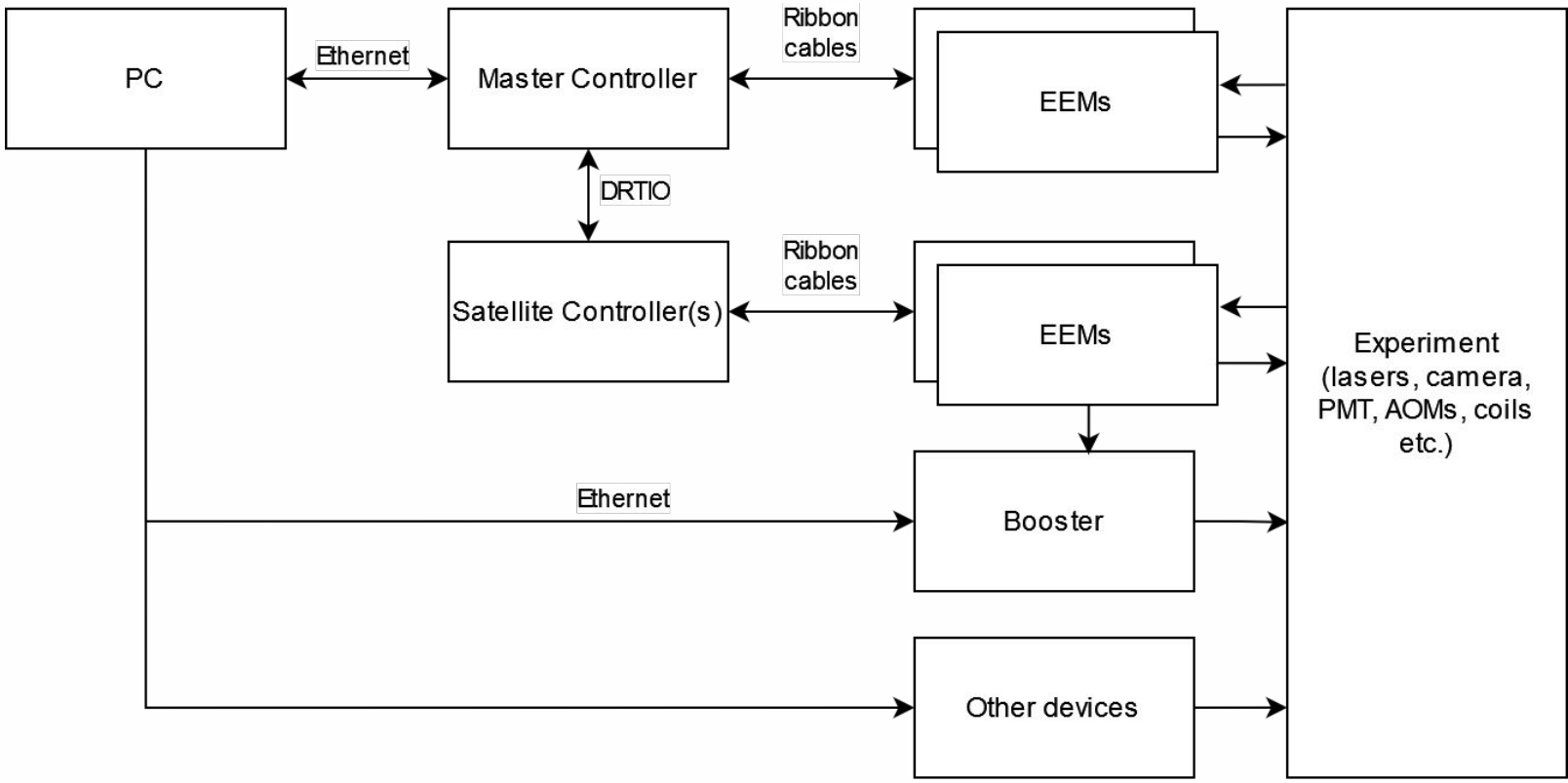
ARTIQ system architecture



What happens with my code?

```
@kernel
def run(self):
    self.core.reset()
    self.ttl1.pulse(100*ns)
```

- Python
- Build stage
- Run stage
  - Access to the FPGA
  - Hard real-time on the FPGA
  - RPC between PC and the FPGA

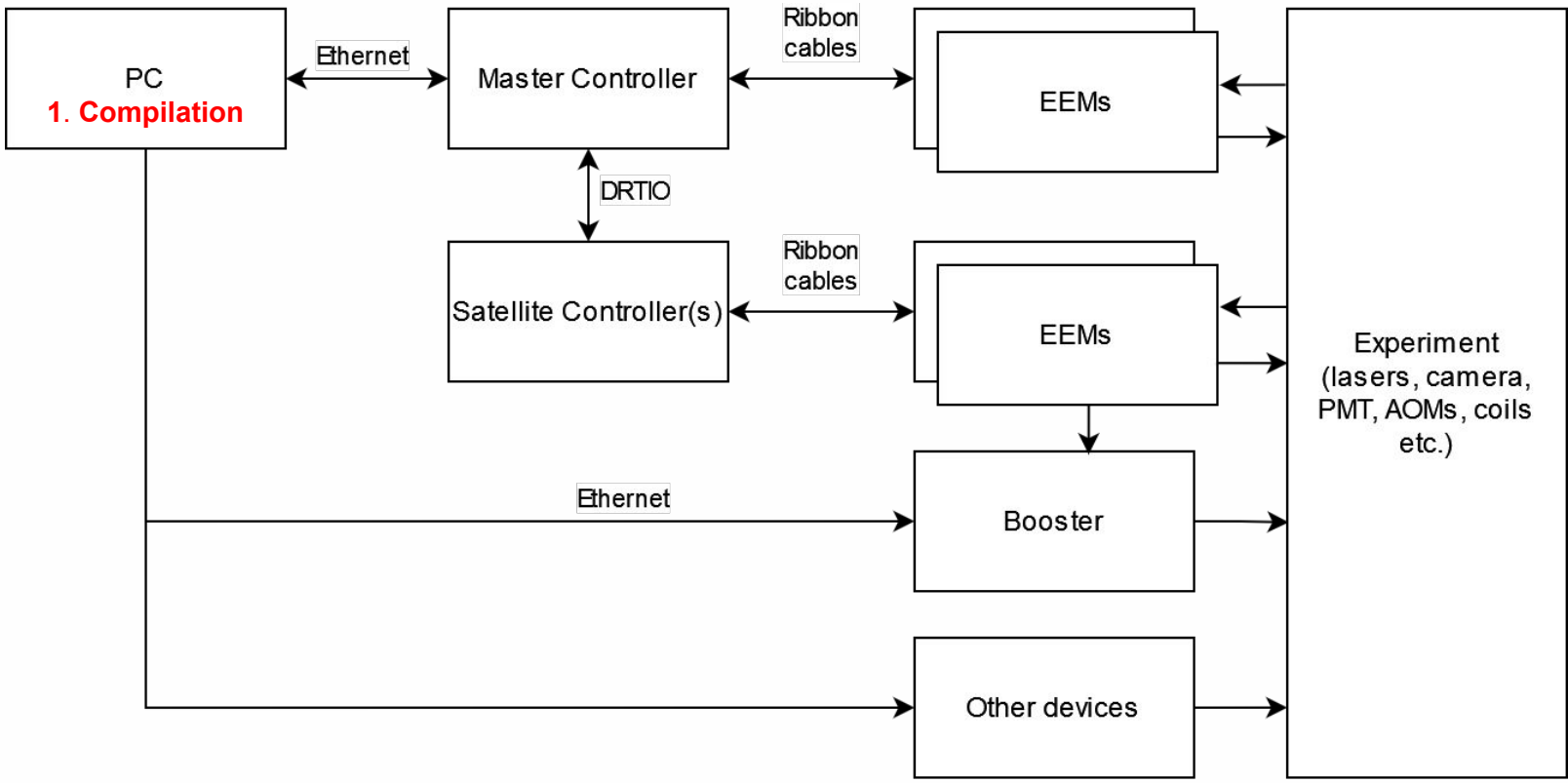


Warsaw University of Technology

What happens with my code?

```
@kernel
def run(self):
    self.core.reset()
    self.ttl1.pulse(100*ns)
```

- Python
- Build stage
- Run stage
  - Access to the FPGA
  - Hard real-time on the FPGA
  - RPC between PC and the FPGA

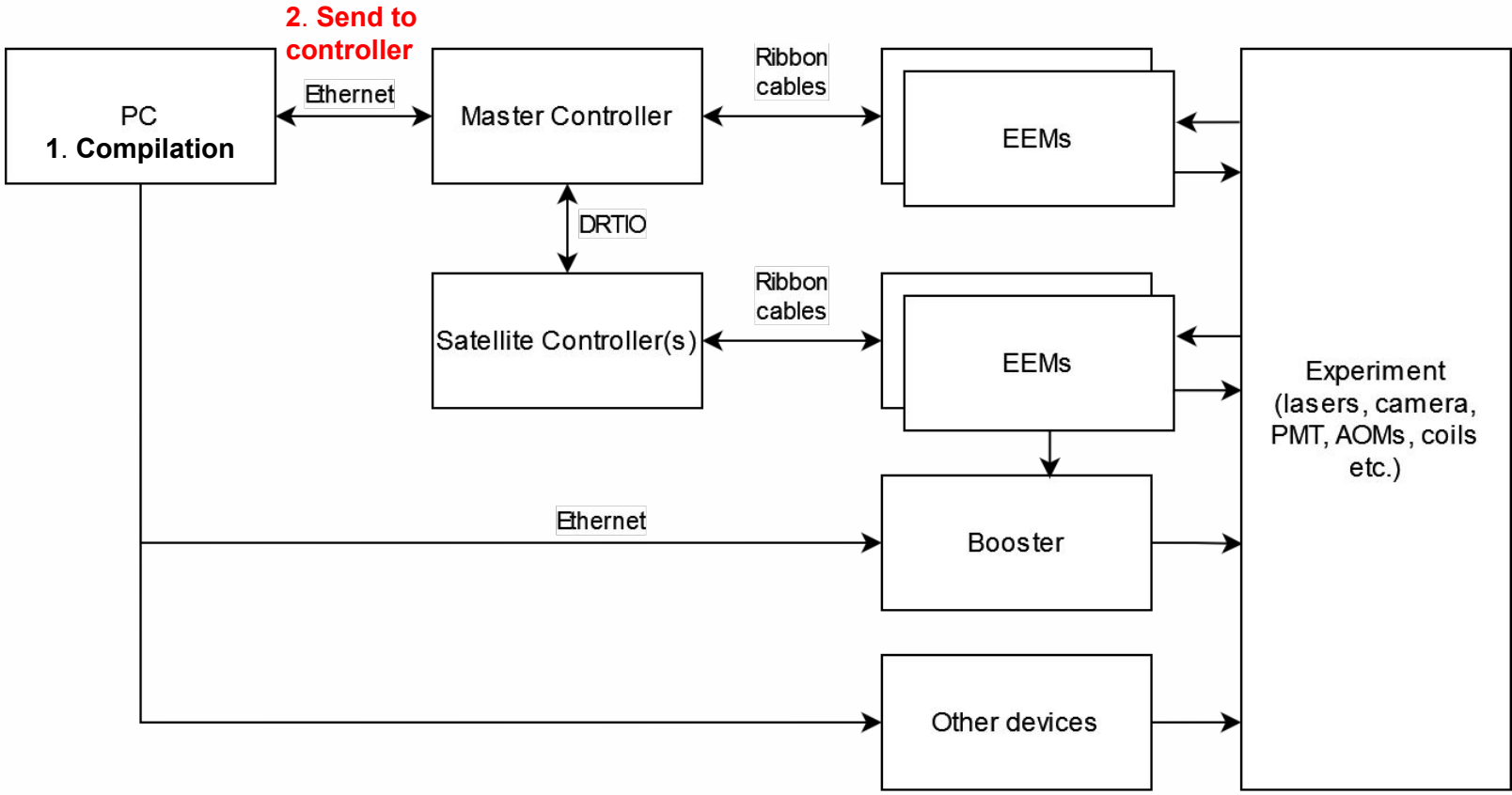


Warsaw University of Technology

What happens with my code?

```
@kernel
def run(self):
    self.core.reset()
    self.ttl1.pulse(100*ns)
```

- Python
- Build stage
- Run stage
  - Access to the FPGA
  - Hard real-time on the FPGA
  - RPC between PC and the FPGA

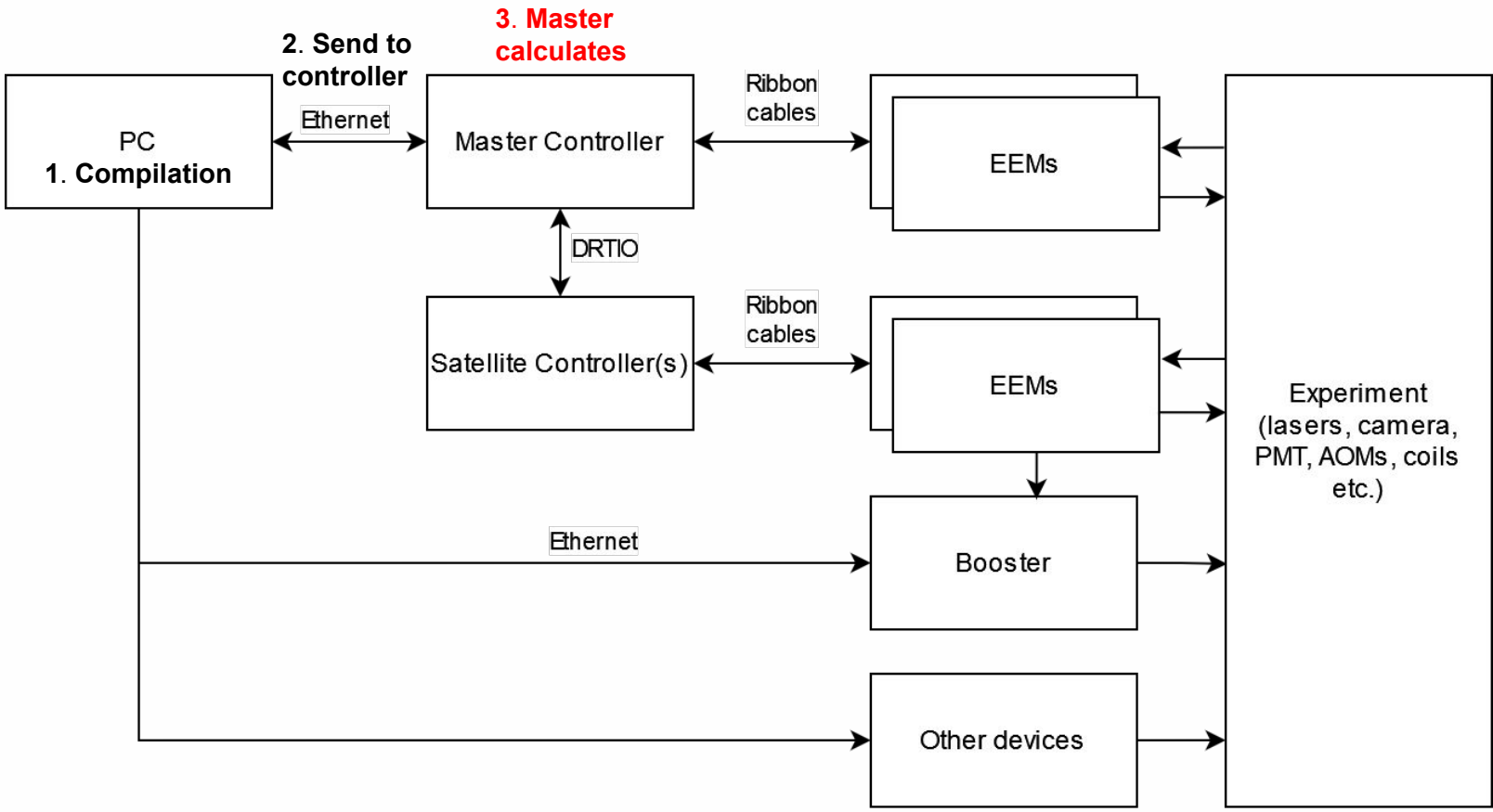


Warsaw University of Technology

What happens with my code?

```
@kernel
def run(self):
    self.core.reset()
    self.ttl1.pulse(100*ns)
```

- Python
- Build stage
- Run stage
  - Access to the FPGA
  - Hard real-time on the FPGA
  - RPC between PC and the FPGA

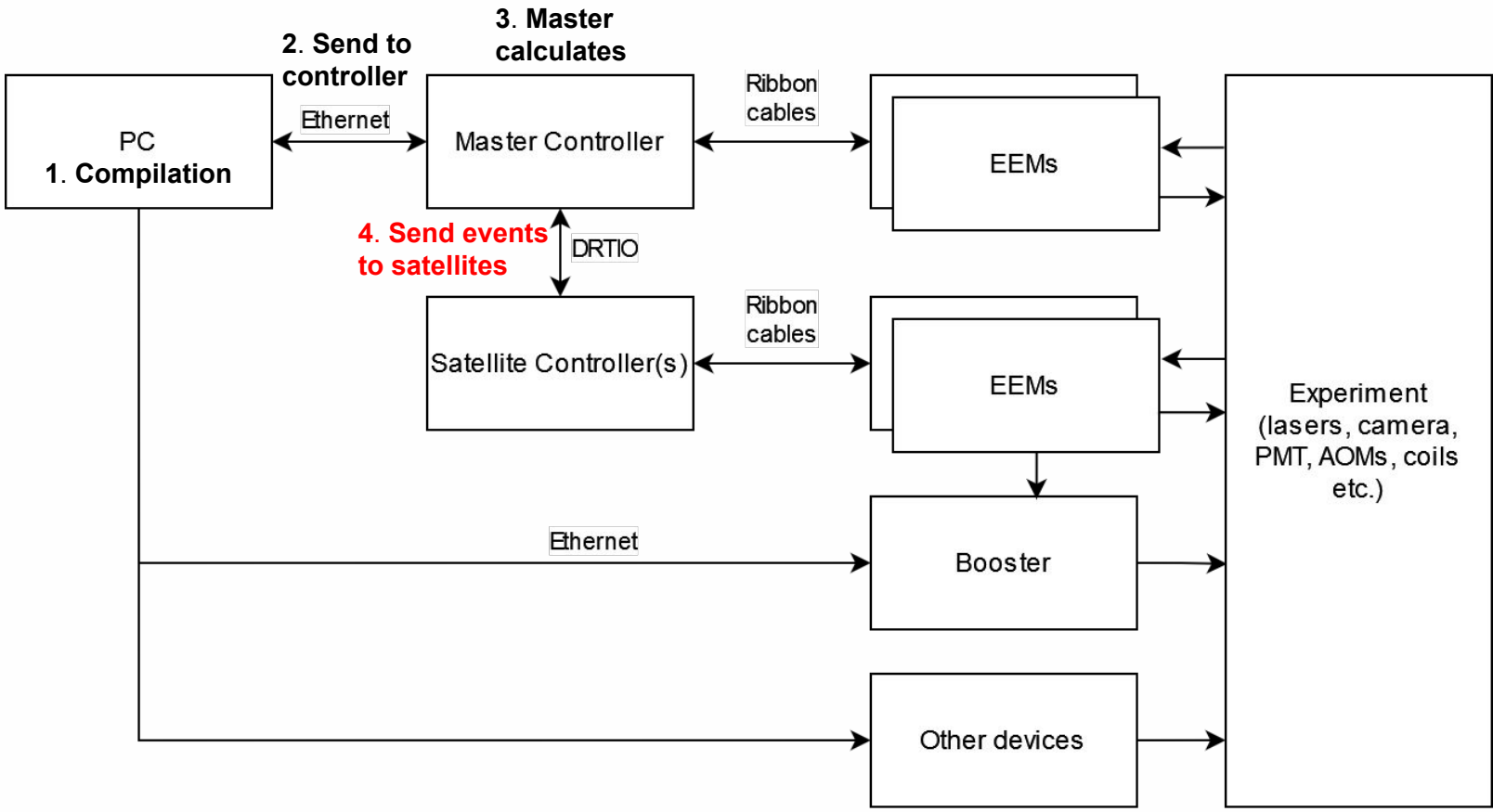


Warsaw University of Technology

What happens with my code?

```
@kernel
def run(self):
    self.core.reset()
    self.ttl1.pulse(100*ns)
```

- Python
- Build stage
- Run stage
  - Access to the FPGA
  - Hard real-time on the FPGA
  - RPC between PC and the FPGA

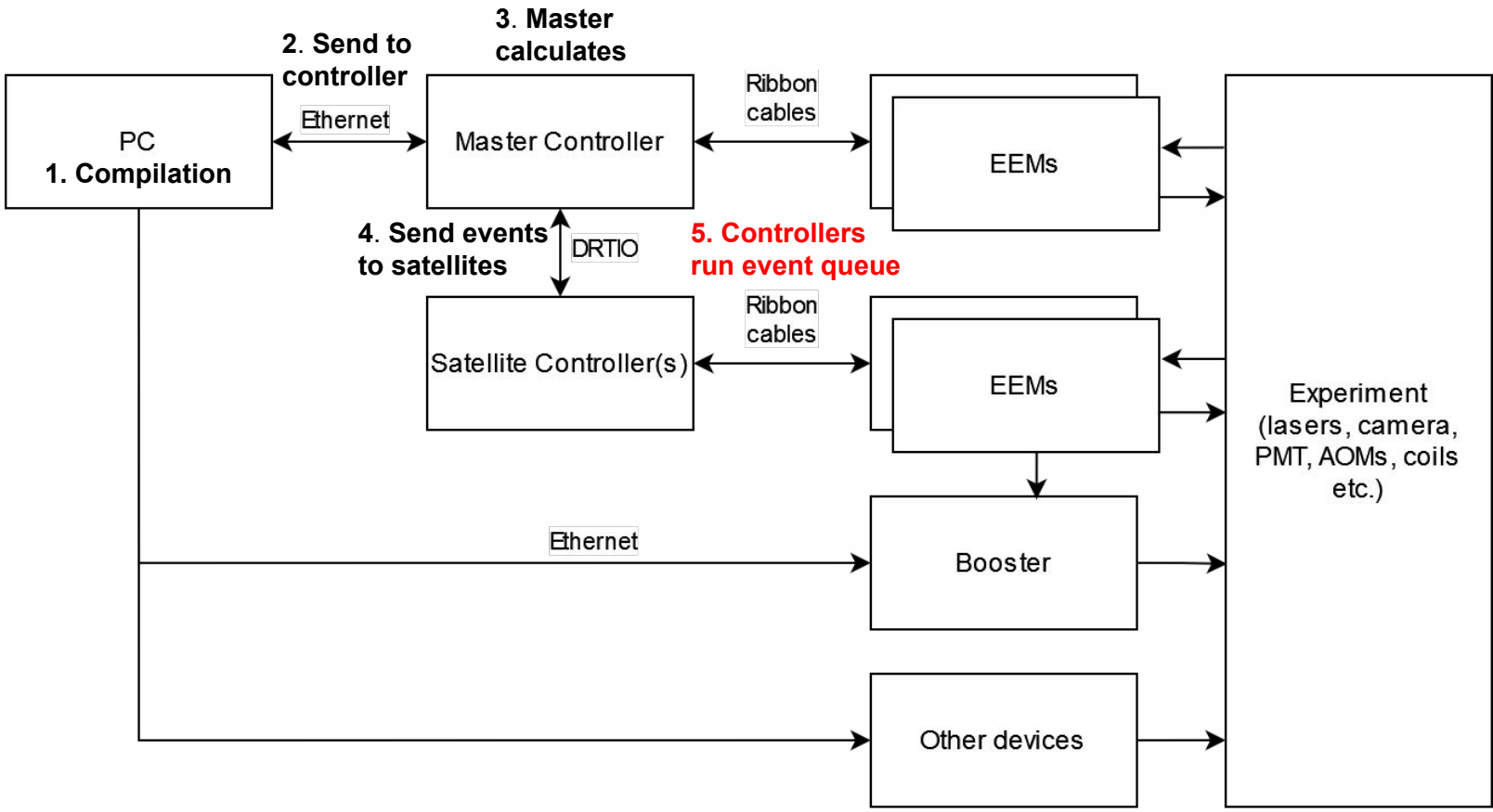


Warsaw University of Technology

What happens with my code?

```
@kernel
def run(self):
    self.core.reset()
    self.ttl1.pulse(100*ns)
```

- Python
- Build stage
- Run stage
  - Access to the FPGA
  - Hard real-time on the FPGA
  - RPC between PC and the FPGA

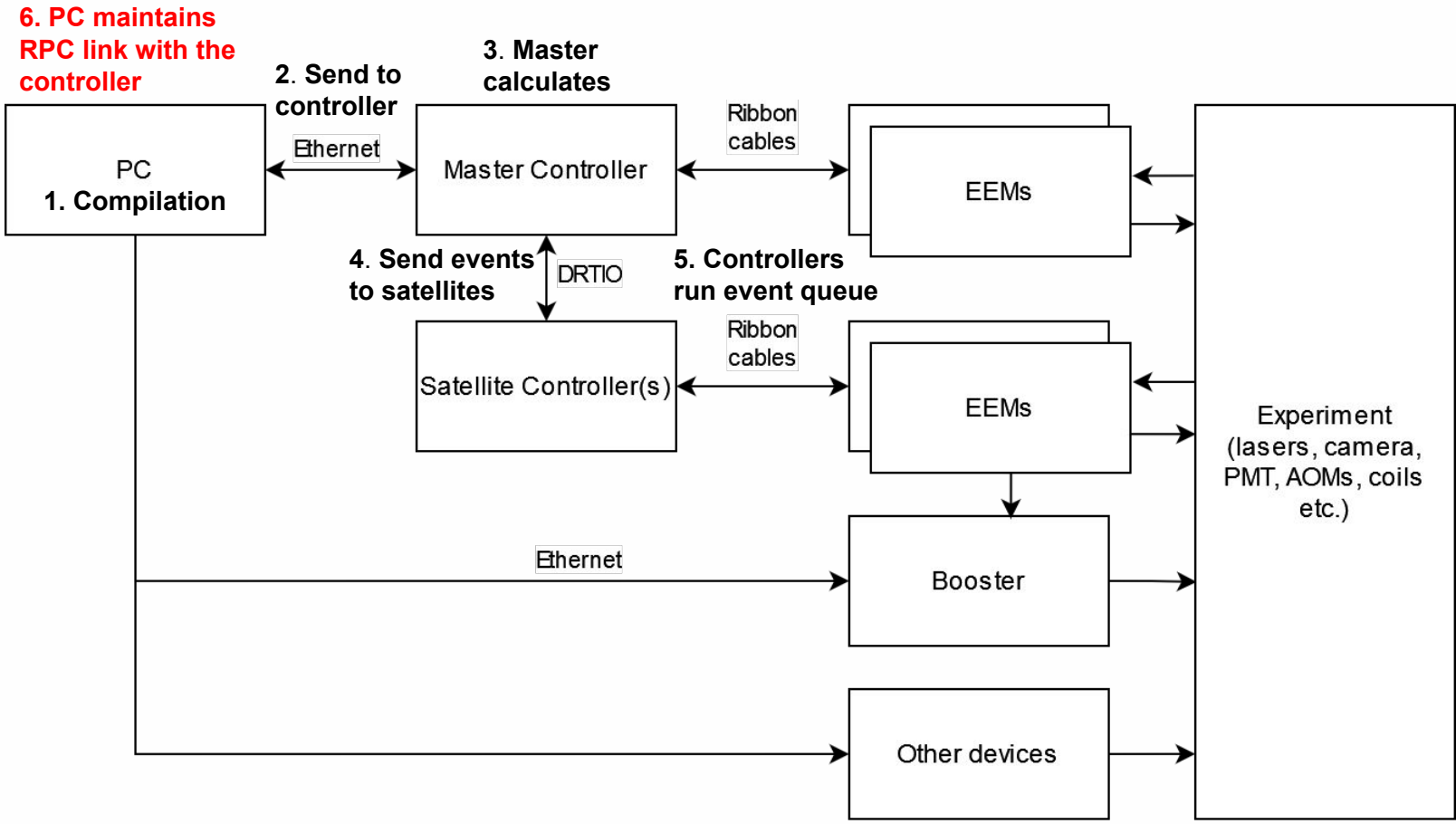


Warsaw University of Technology

What happens with my code?

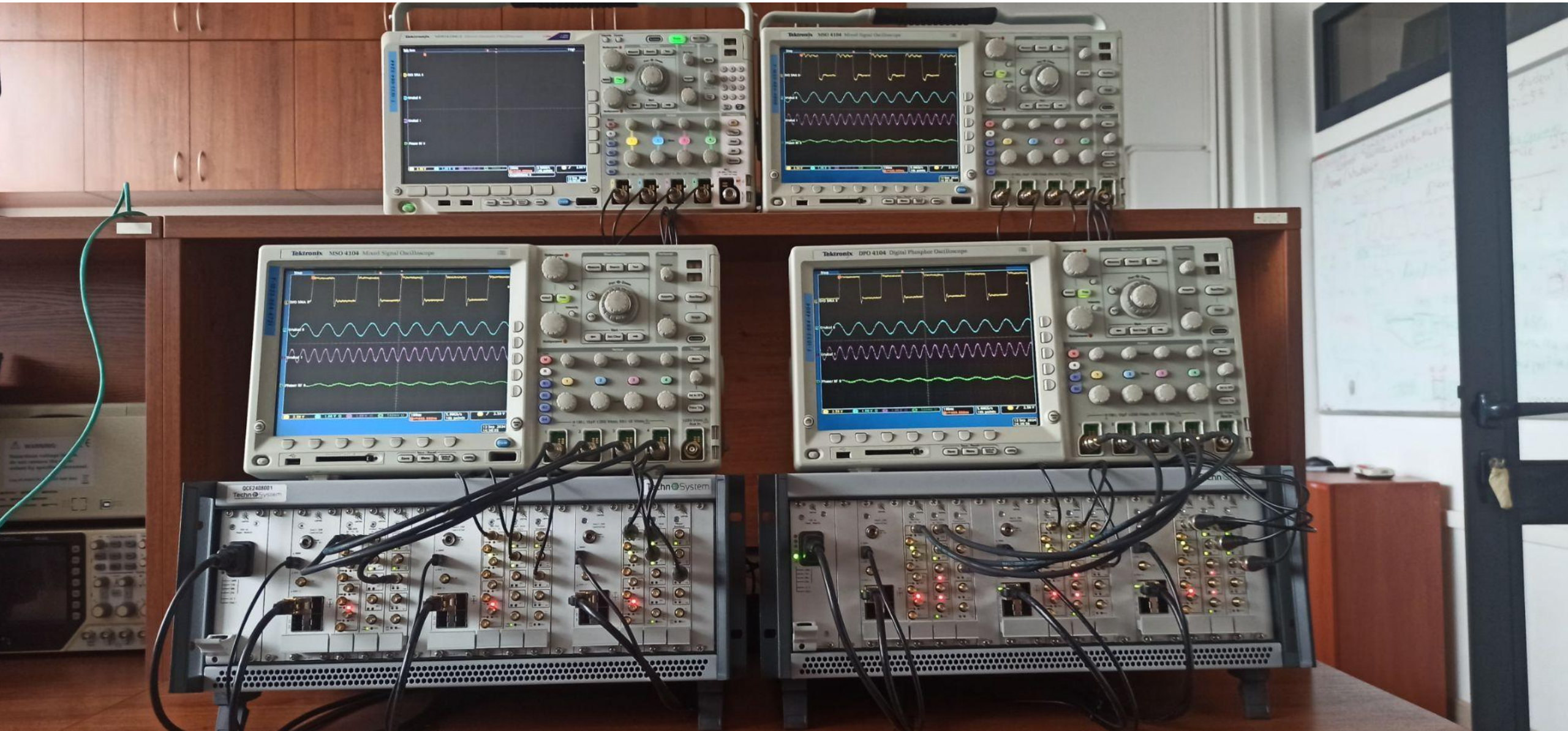
```
@kernel
def run(self):
    self.core.reset()
    self.ttl1.pulse(100*ns)
```

- Python
- Build stage
- Run stage
  - Access to the FPGA
  - Hard real-time on the FPGA
  - RPC between PC and the FPGA

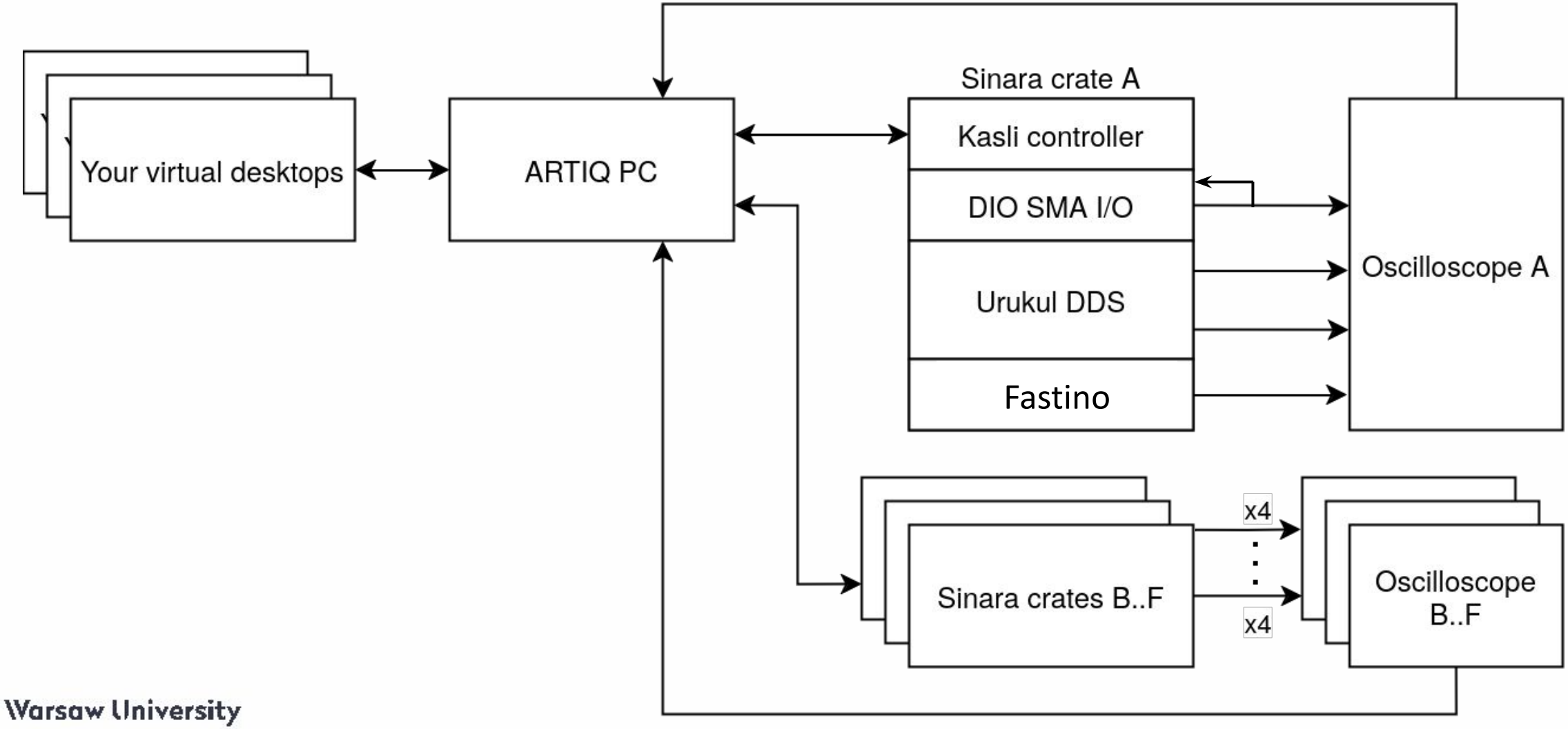


Warsaw University of Technology

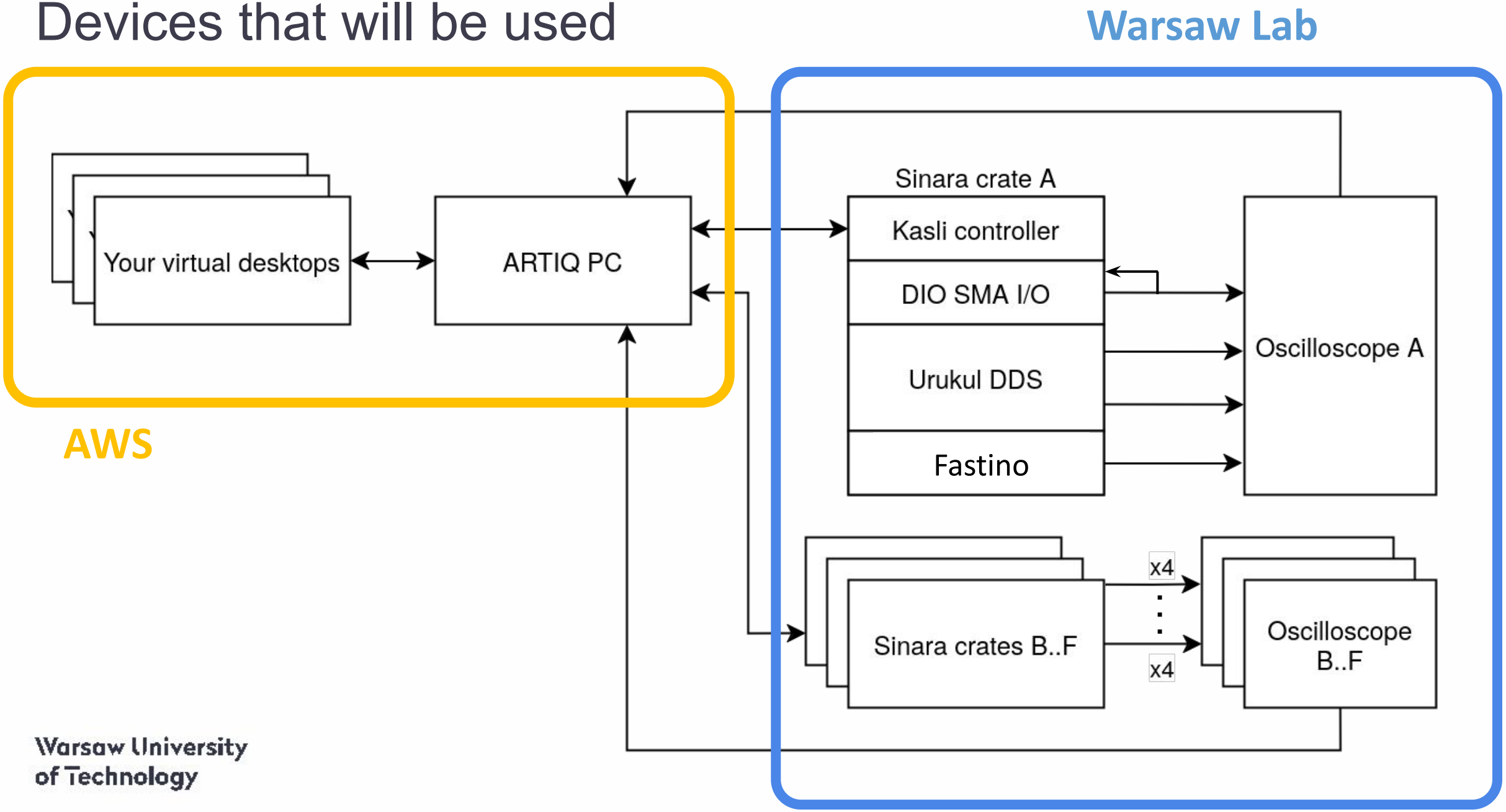
Devices that will be used



Devices that will be used



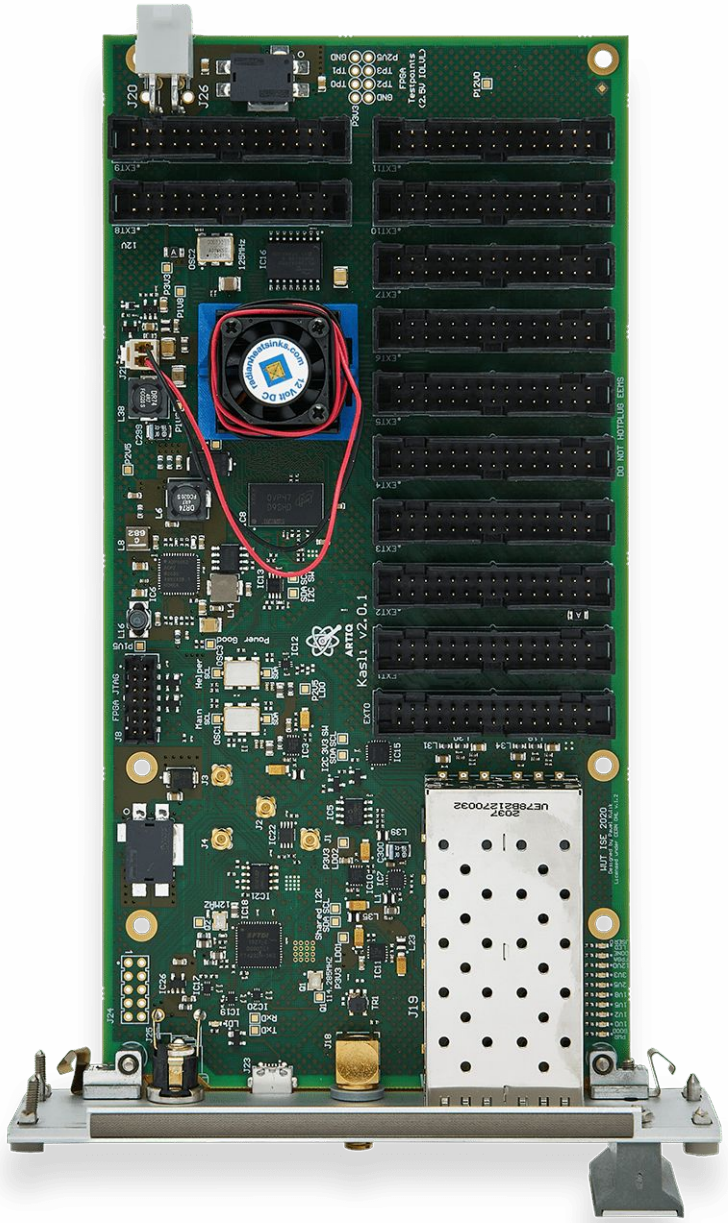
Devices that will be used



Warsaw University of Technology

Devices that will be used

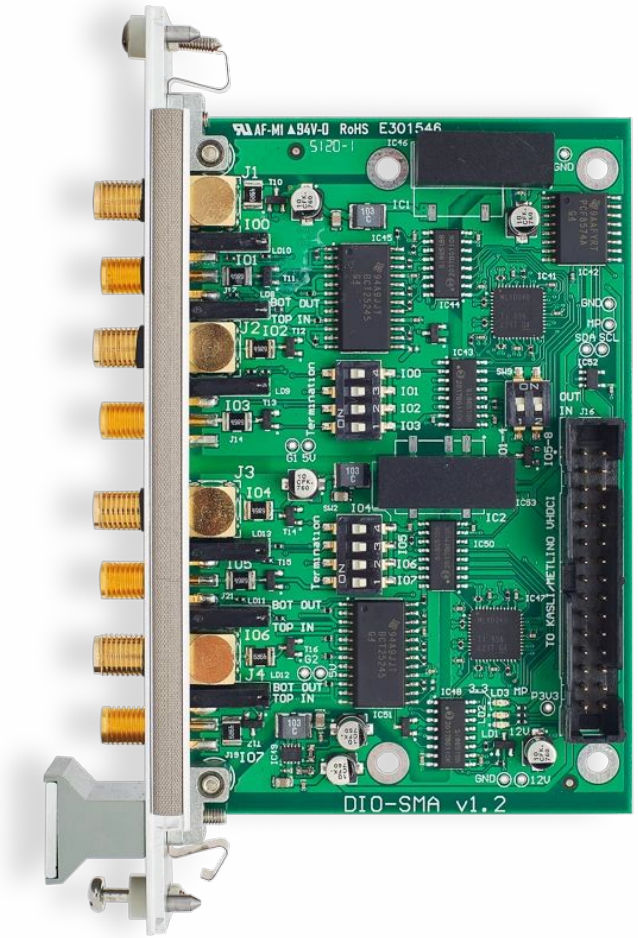
- Kasli
  - AMD Artix-7 based controller
  - up to 12 peripherals
  - up to 3 downstream satellites
  - can be both master and satellite
  - https://github.com/sinara-hw/Kasli



Warsaw University of Technology

Devices that will be used

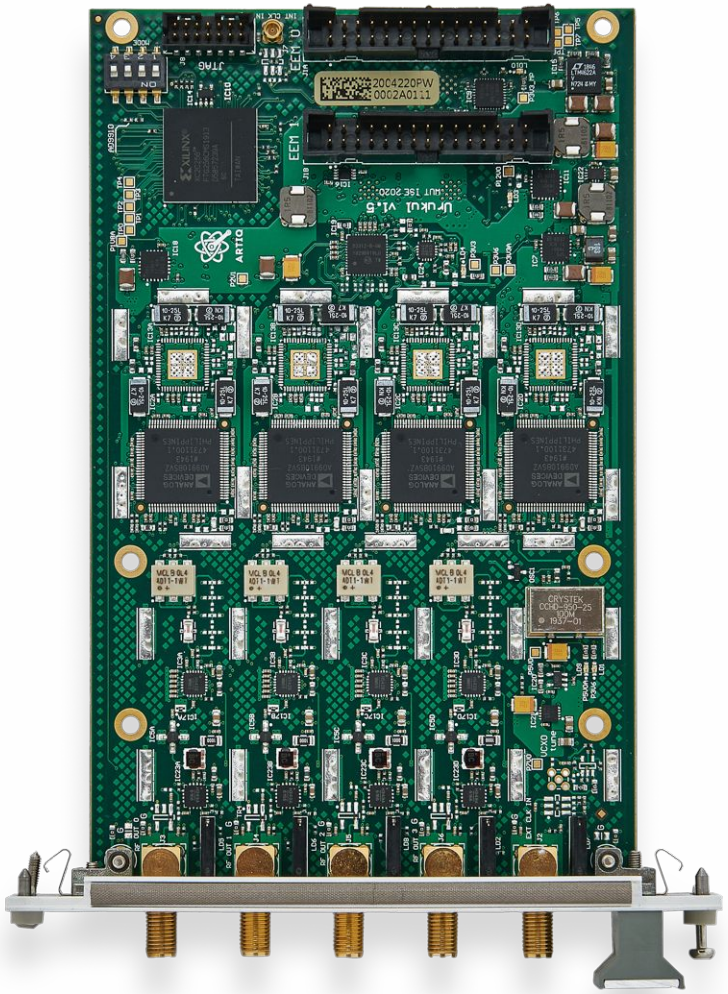
- Kasli
- DIO TTL
  - 8 digital IOs channels
  - selectable 50 Ohm termination
  - min. pulse width 5 ns
  - https://github.com/sinara-hw/DIO_SMA/wiki



Warsaw University of Technology

Devices that will be used

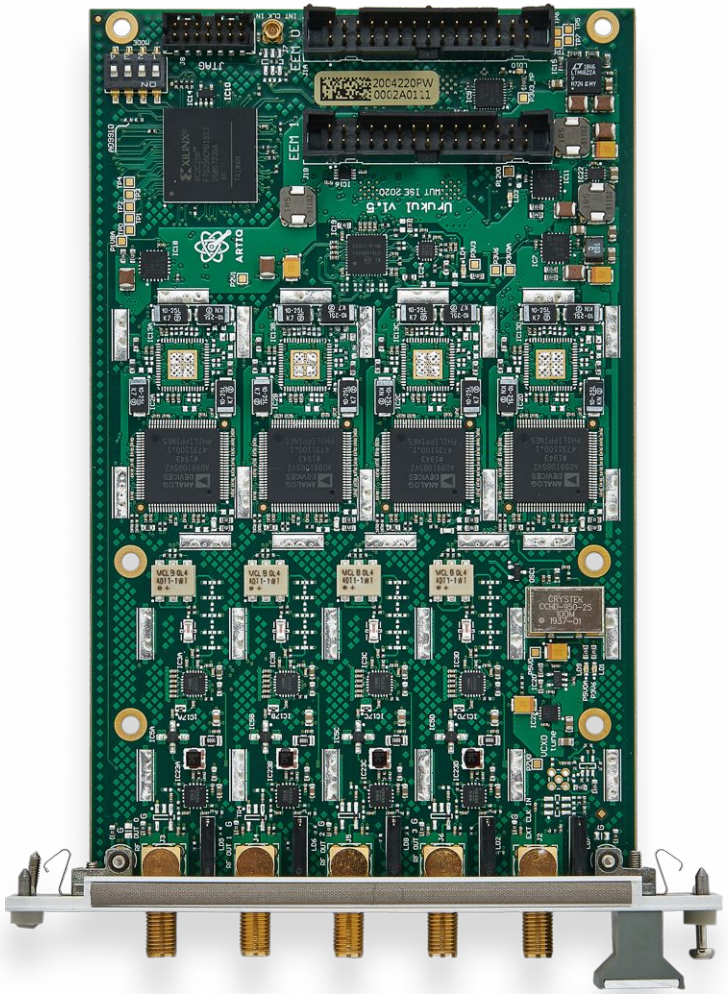
- Kasli
- DIO TTL
- Urukul DDS
  - 4 channels GS/s DDS
  - frequency up to 400 MHz
  - ~0.25 Hz resolution
  - https://github.com/sinara-hw/Urukul/wiki



Warsaw University of Technology

Devices that will be used

- Kasli
- DIO TTL
- Urukul DDS: 3 independent “devices”
  - DDS
  - Attenuator
  - Output switch (on/off output signal)



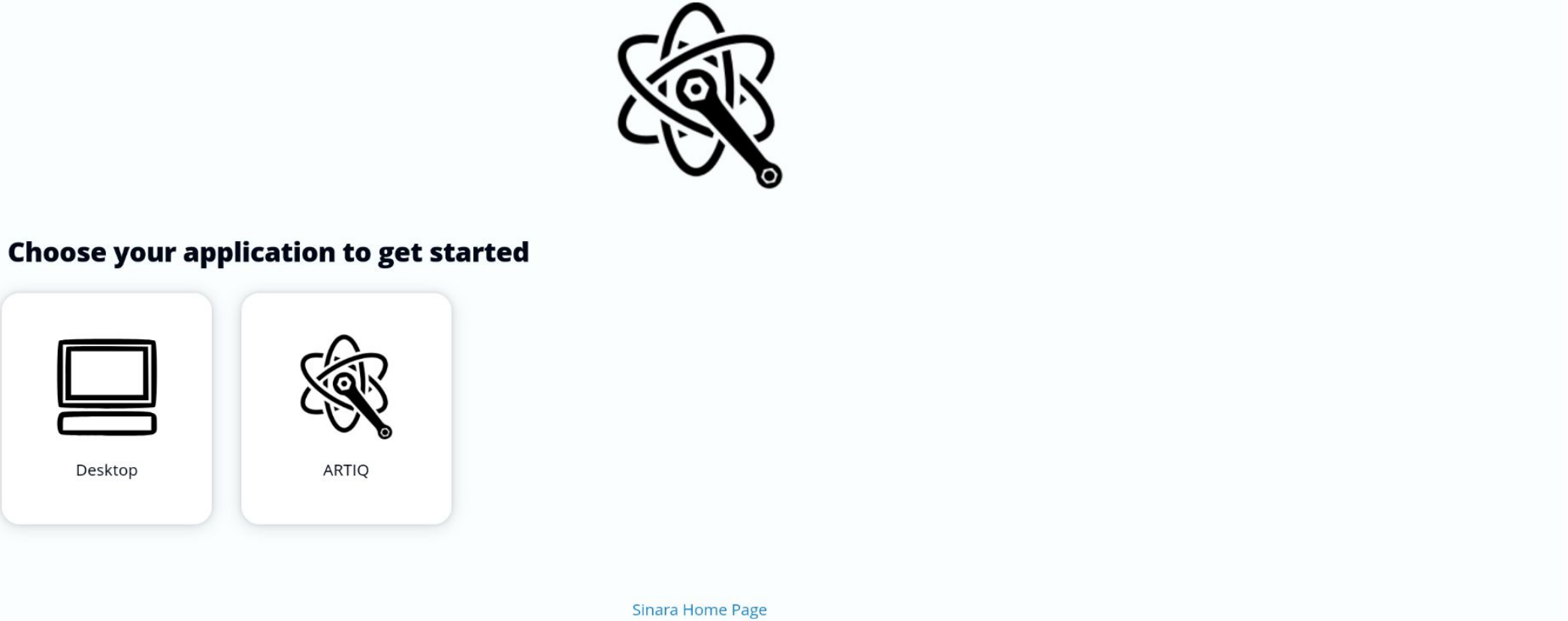
Warsaw University of Technology

Organization

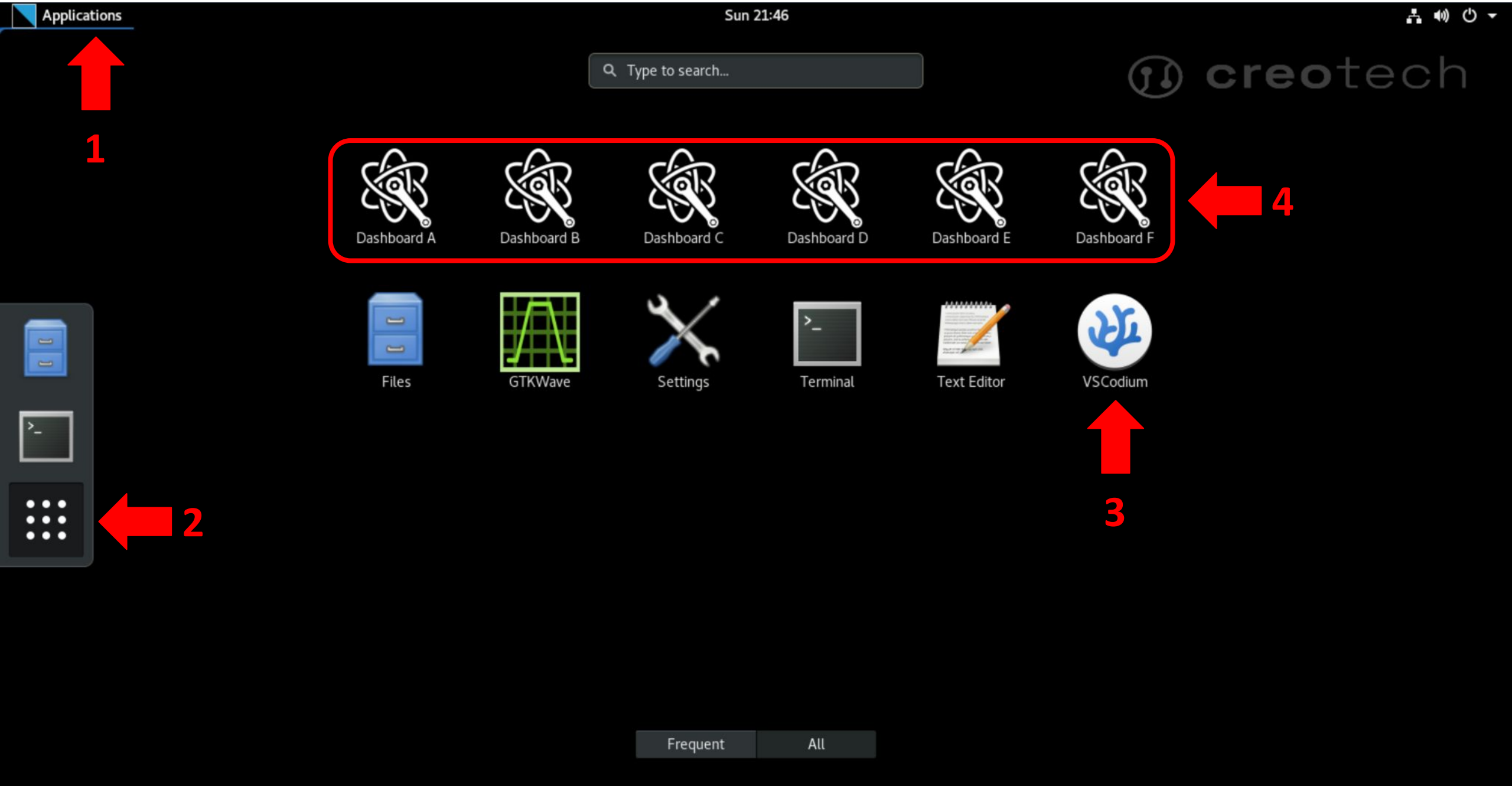
Connect to virtual desktop using your personal link

Organization

1. Connect to virtual desktop using your personal link



Organization



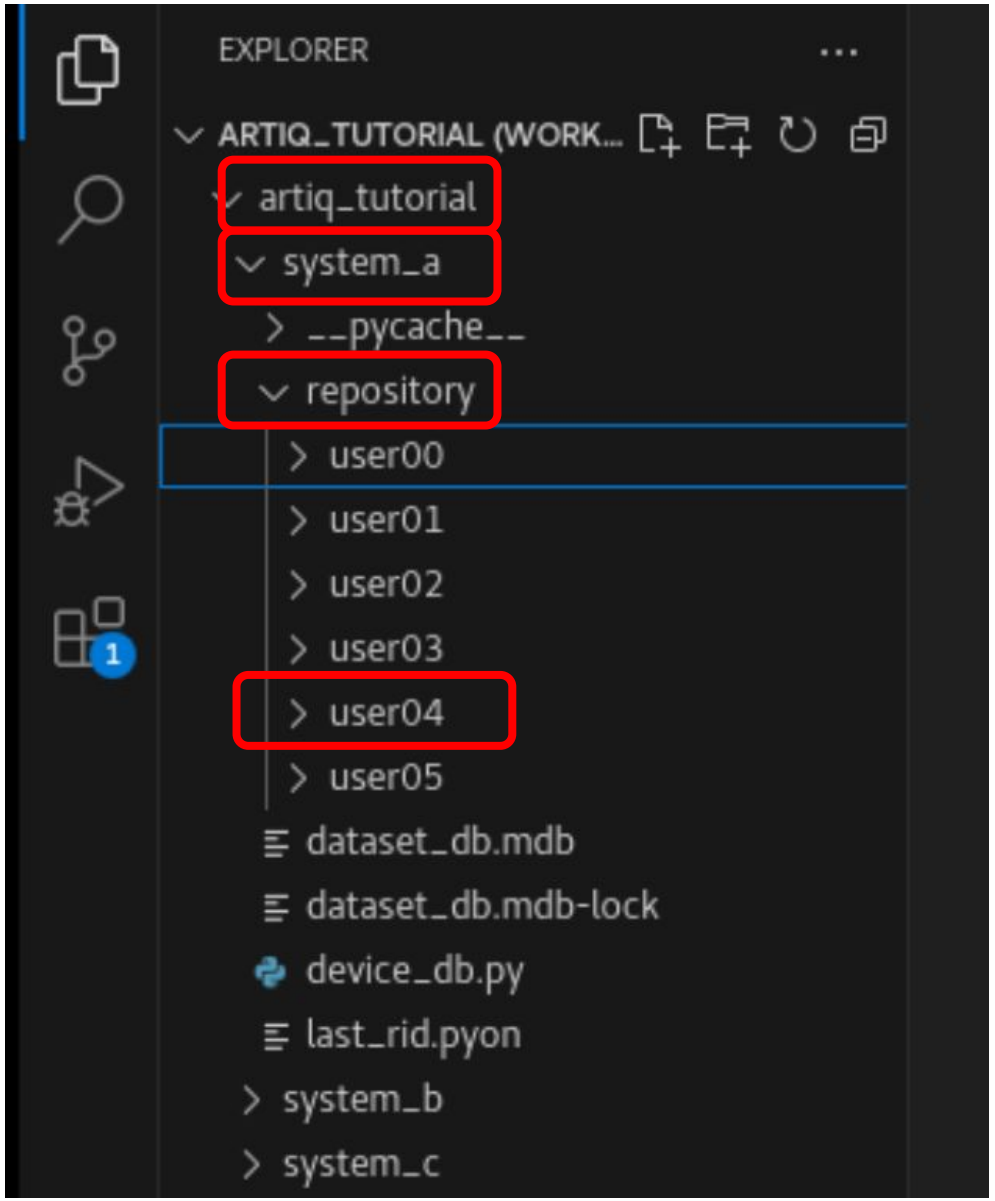
Organization

artiq_tutorial

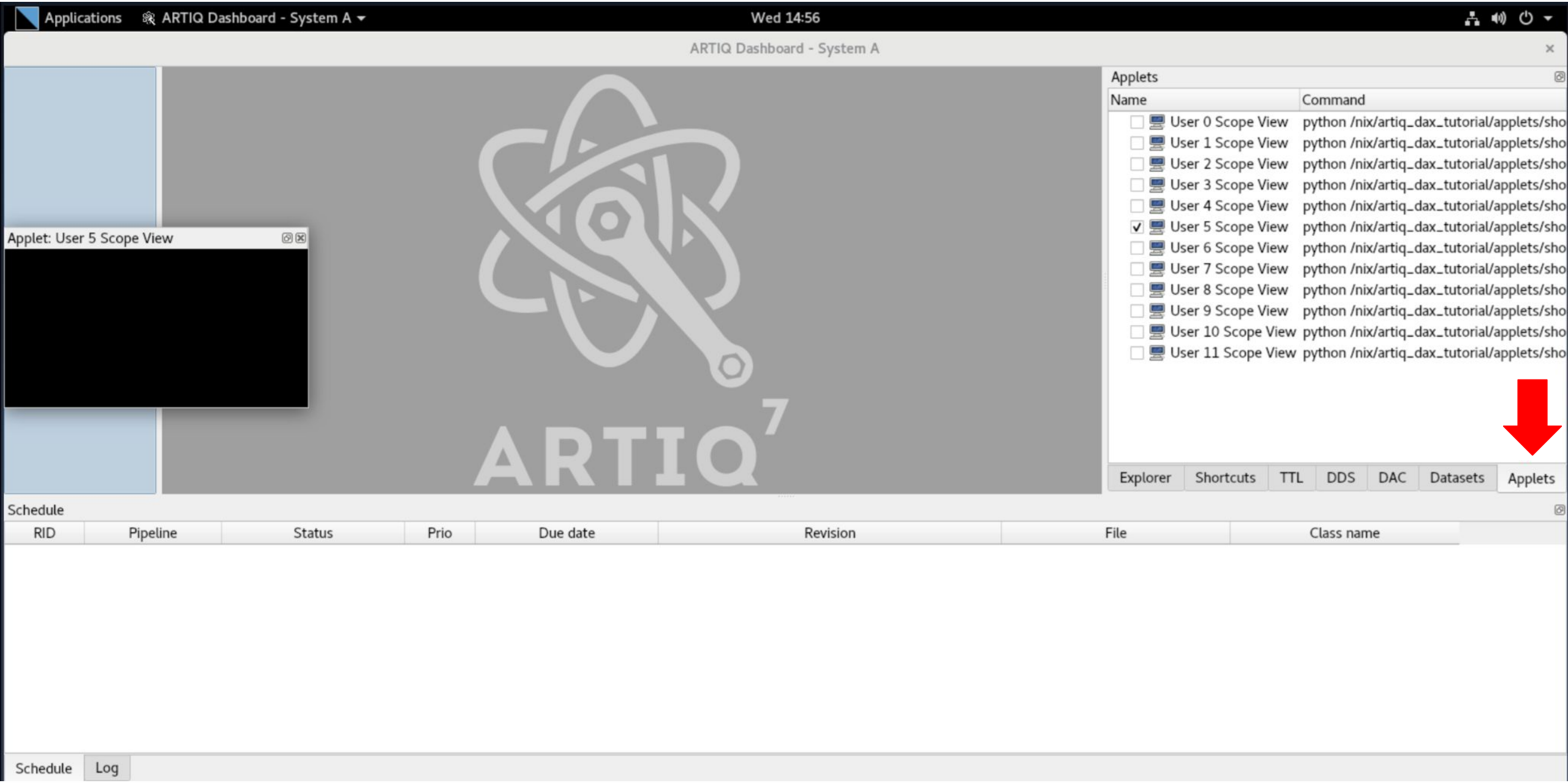
system_<a..f>

repository

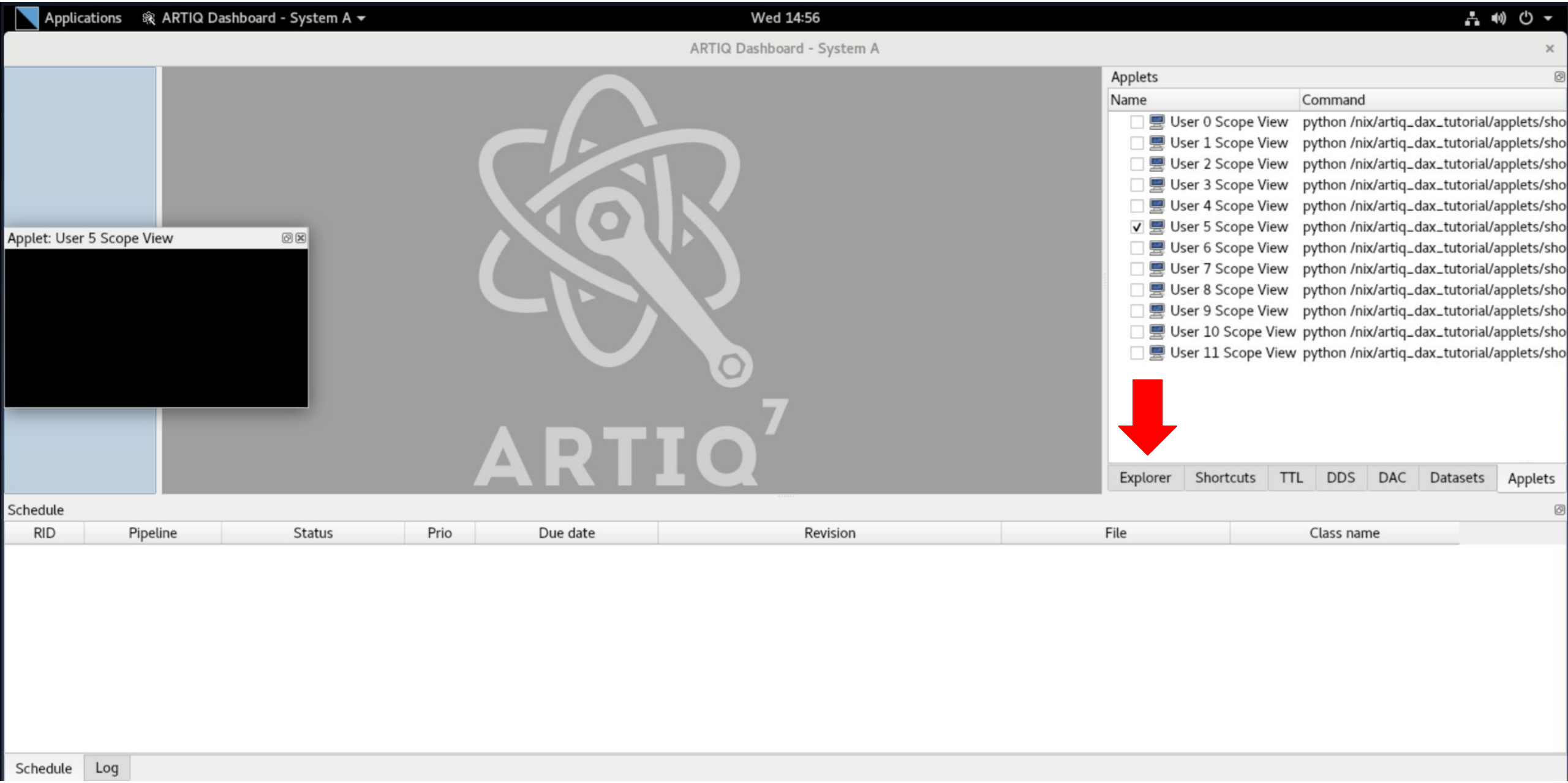
user<00..05>



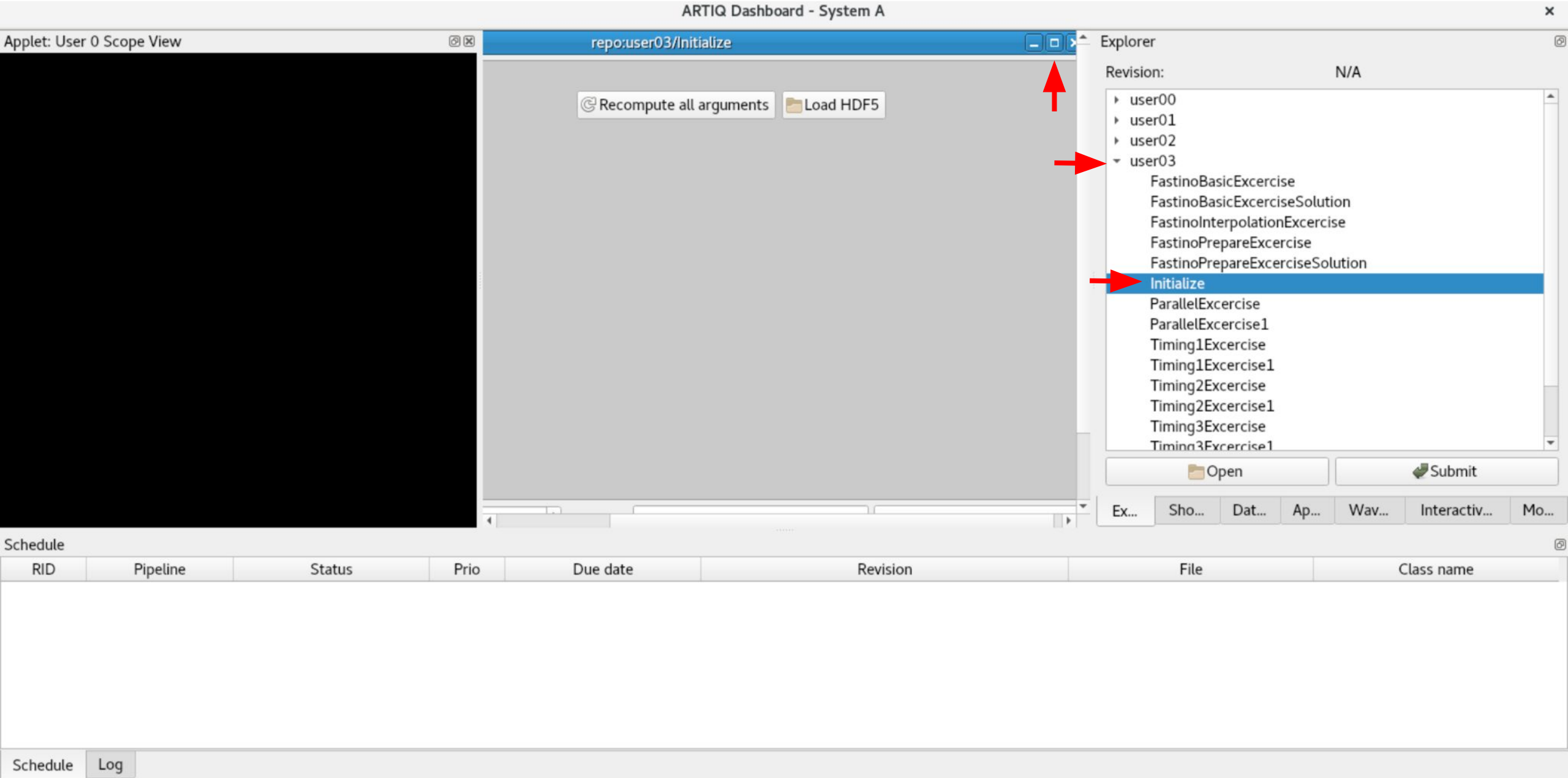
Organization



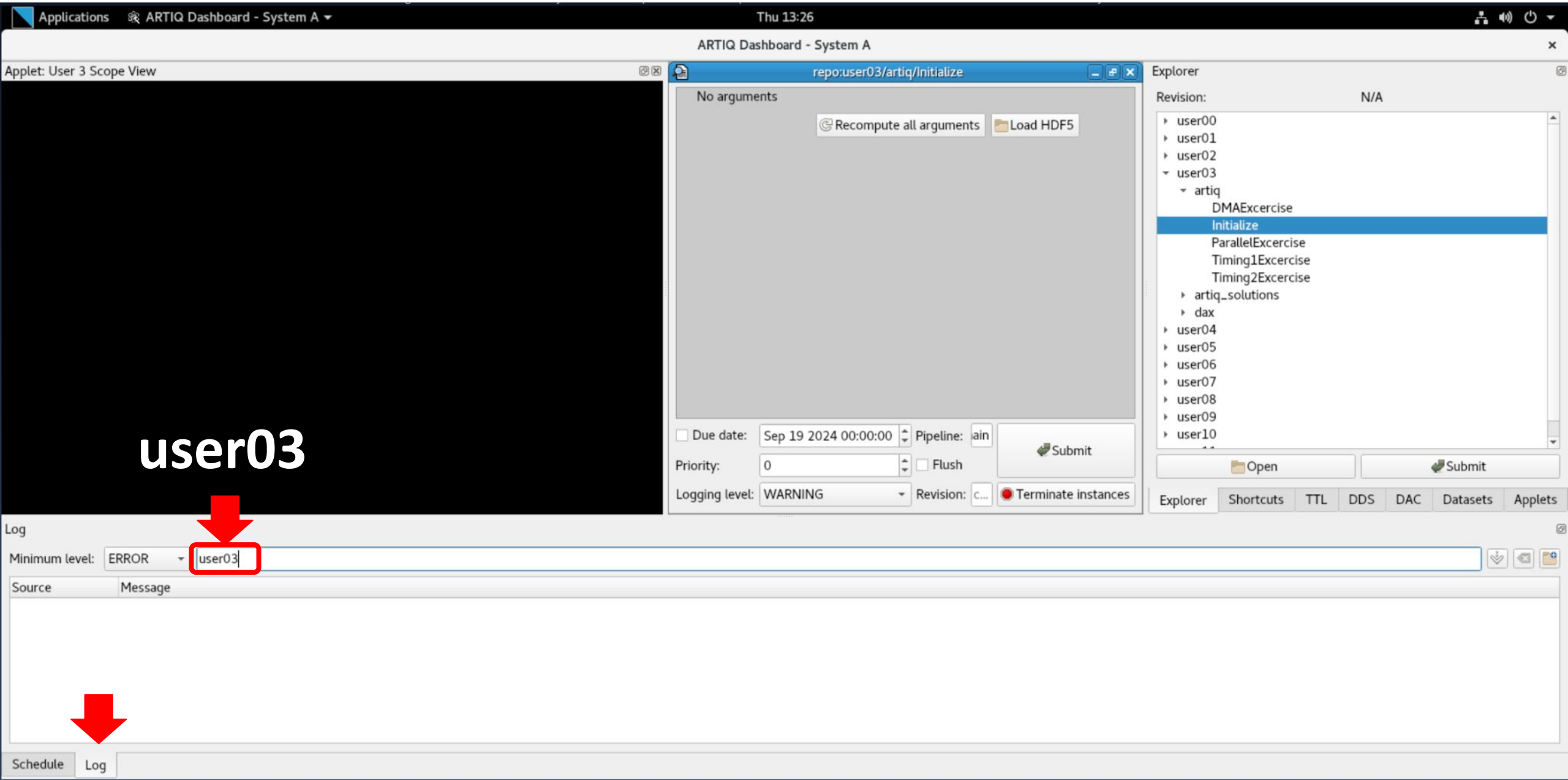
Organization



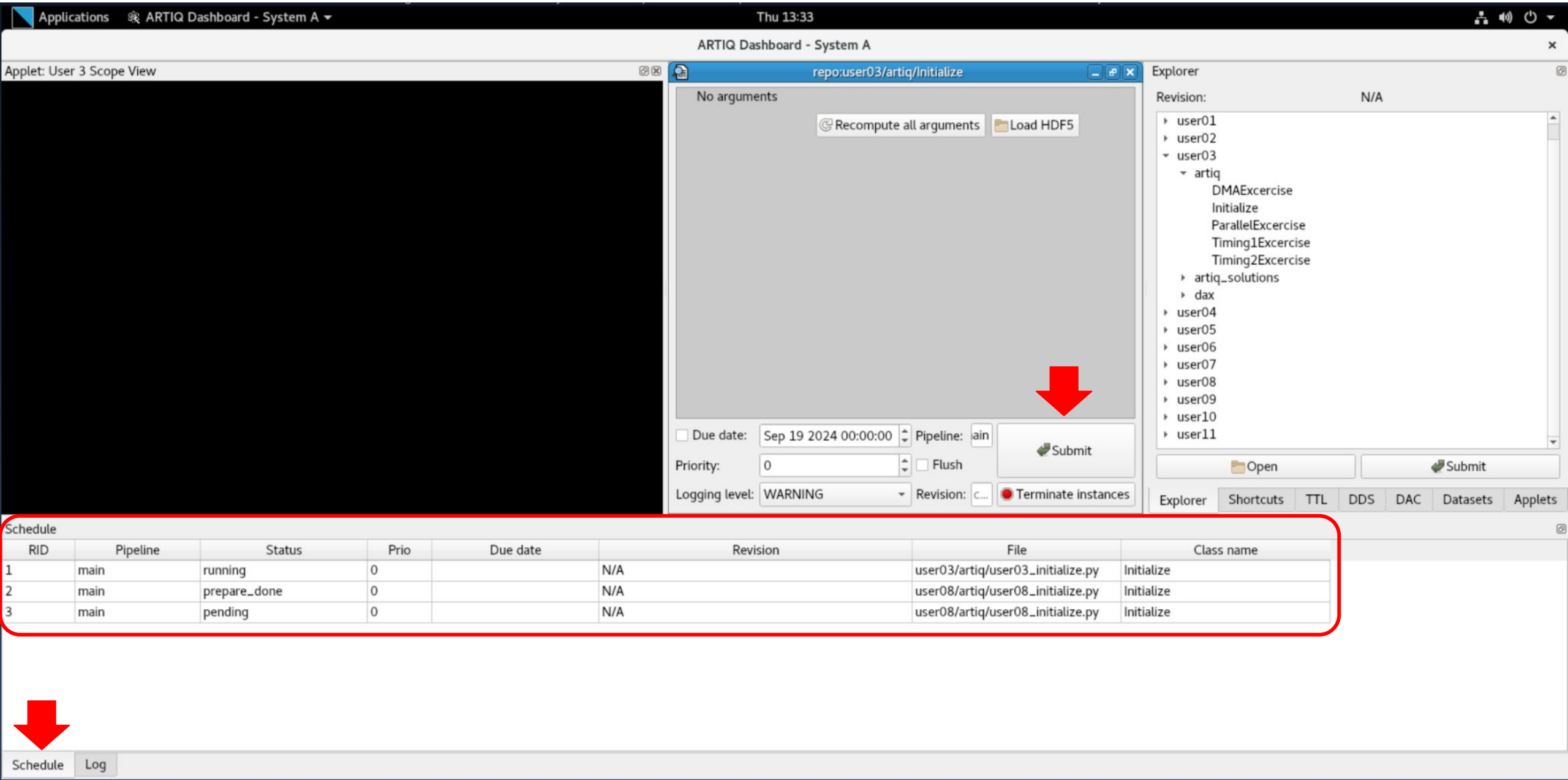
Organization



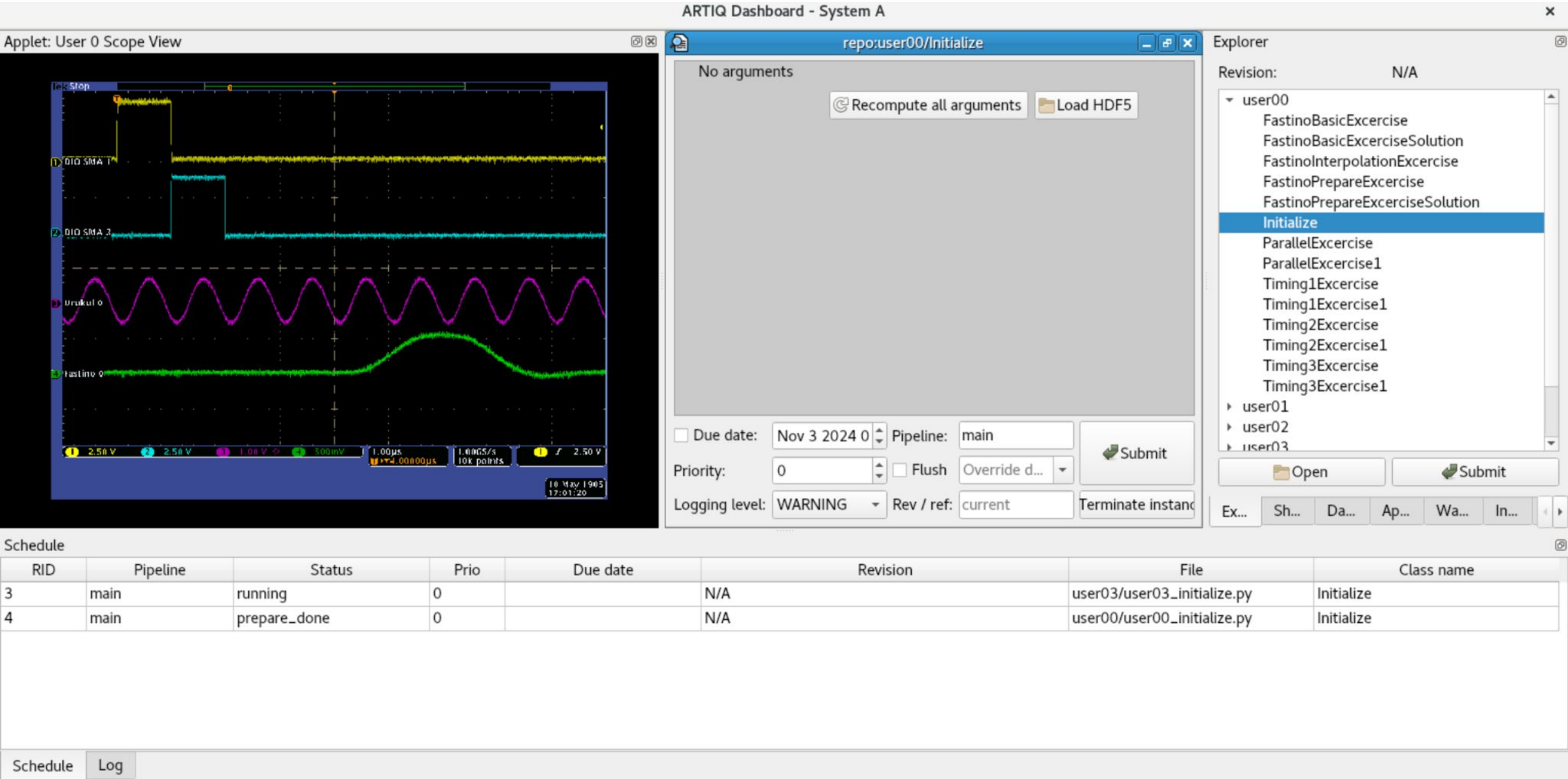
Organization



Organization

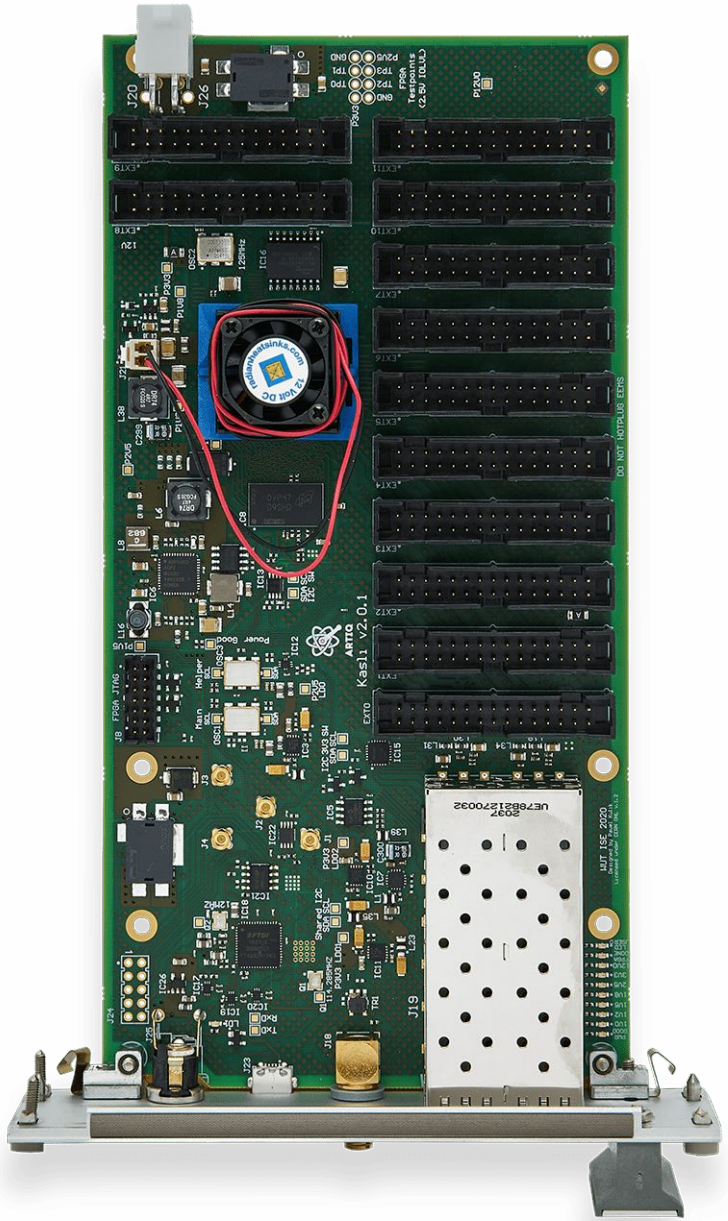


Organization



ARTIQ DSL - what can be done in FPGA?

- Subset of Python:
  - Objects
  - Conditionals (if..else structure)
  - Loops, iterating over lists
  - Exceptions
  - Code management

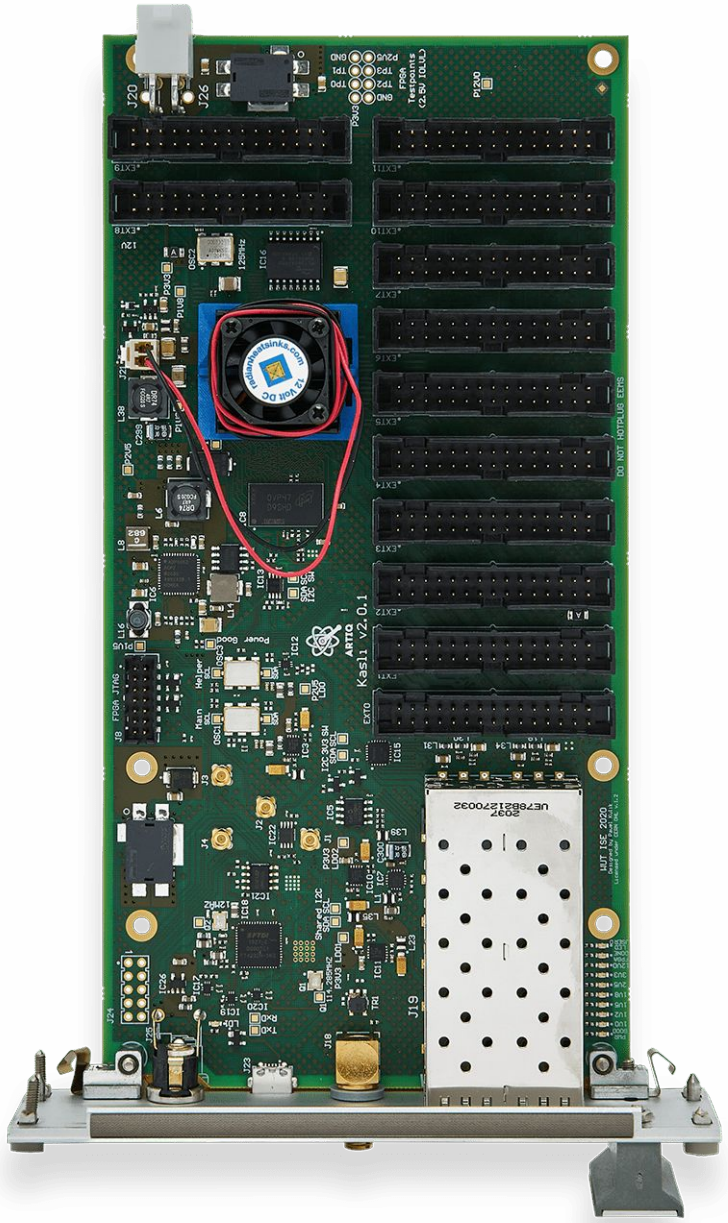


More detailed description: m-labs.hk/artiq/manual/compiler.html

Warsaw University of Technology

ARTIQ DSL - what can be done in FPGA?

- Subset of Python:
  - Objects
  - Conditionals (if..else structure)
  - Loops, iterating over lists
  - Exceptions
  - Code management
- System specific:
  - Timing functions
  - Parallel / Sequential blocks
  - DMA
  - RPC



More detailed description: m-labs.hk/artiq/manual/compiler.html

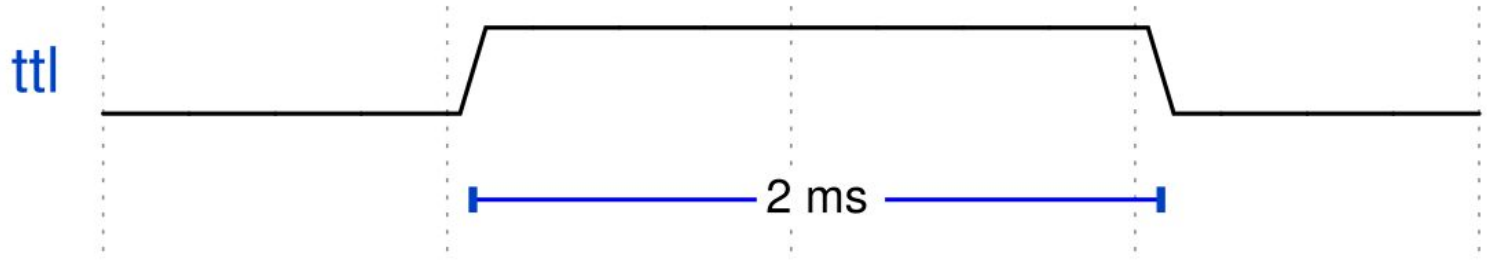
Warsaw University of Technology

ARTIQ real-time concepts

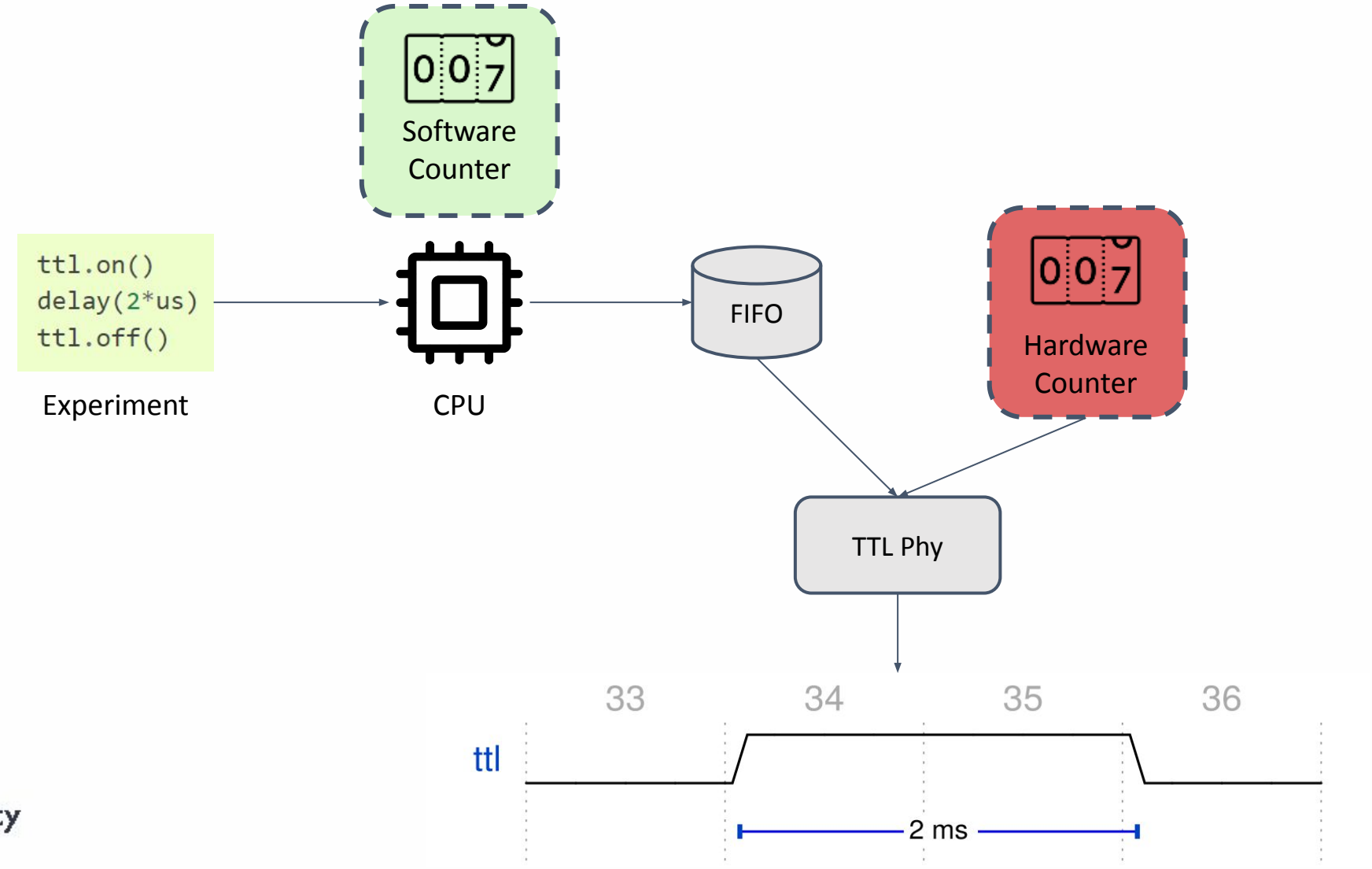
```
ttl.on()
delay(2*us)
ttl.off()
```

ARTIQ real-time concepts

```
ttl.on()
delay(2*us)
ttl.off()
```

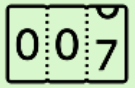


ARTIQ real-time concepts



Warsaw University of Technology

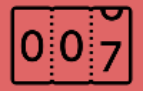
ARTIQ real-time concepts



Software Counter

Timeline cursor (now)

Altered by experiment's commands

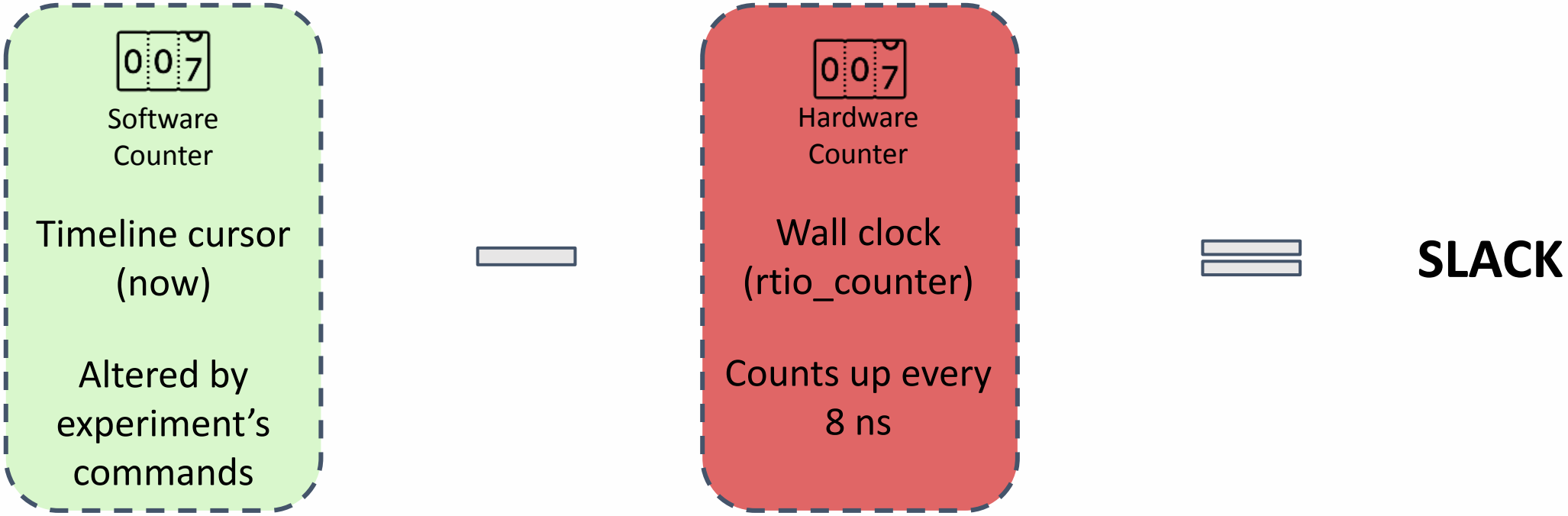


Hardware Counter

Wall clock (rtio_counter)

Counts up every 8 ns

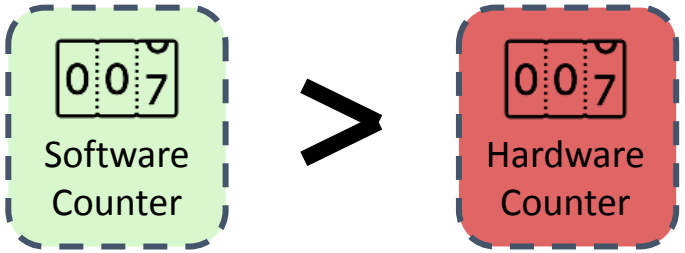
ARTIQ real-time concepts



ARTIQ real-time concepts

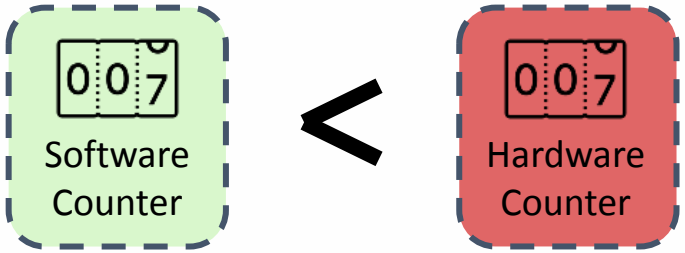
SLACK

POSITIVE



Planning the future

NEGATIVE



Altering the past

RTIOUnderflowException

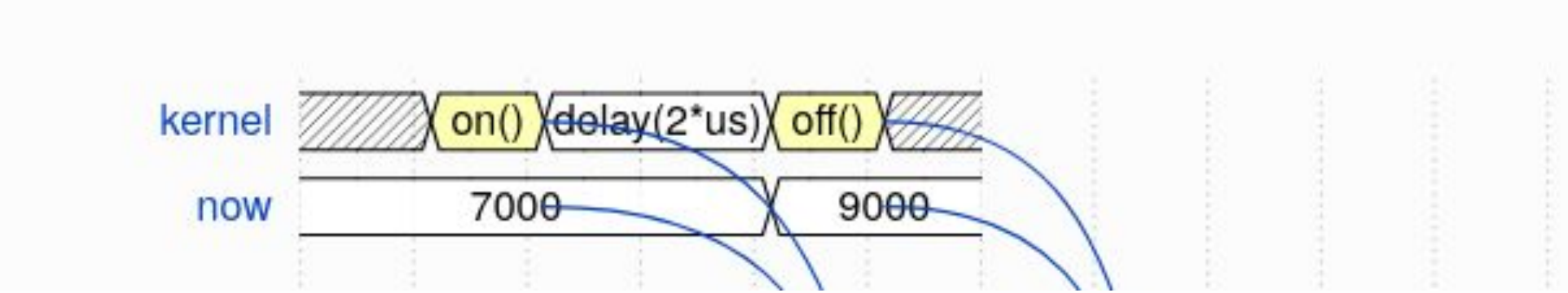
Warsaw University of Technology

ARTIQ real-time concepts

ARTIQ experiment execution

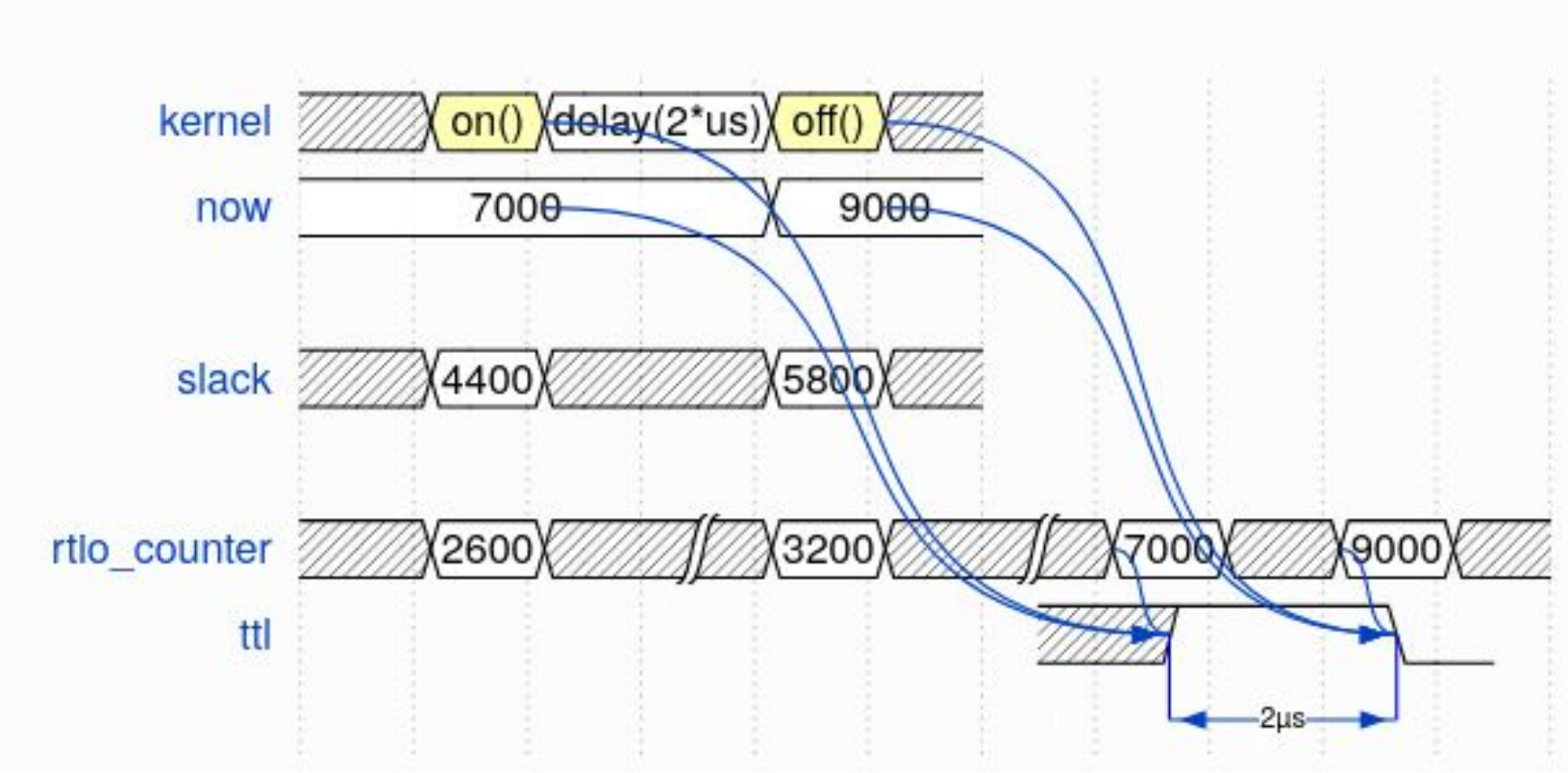
ARTIQ real-time concepts

```
ttl.on()
delay(2*us)
ttl.off()
```



ARTIQ real-time concepts

```
ttl.on()
delay(2*us)
ttl.off()
```

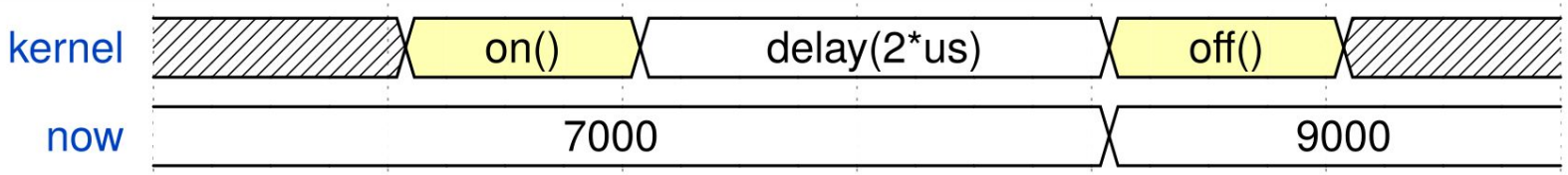


ARTIQ real-time concepts

Manipulating timeline

Delay:

```
delay(2*us)
```

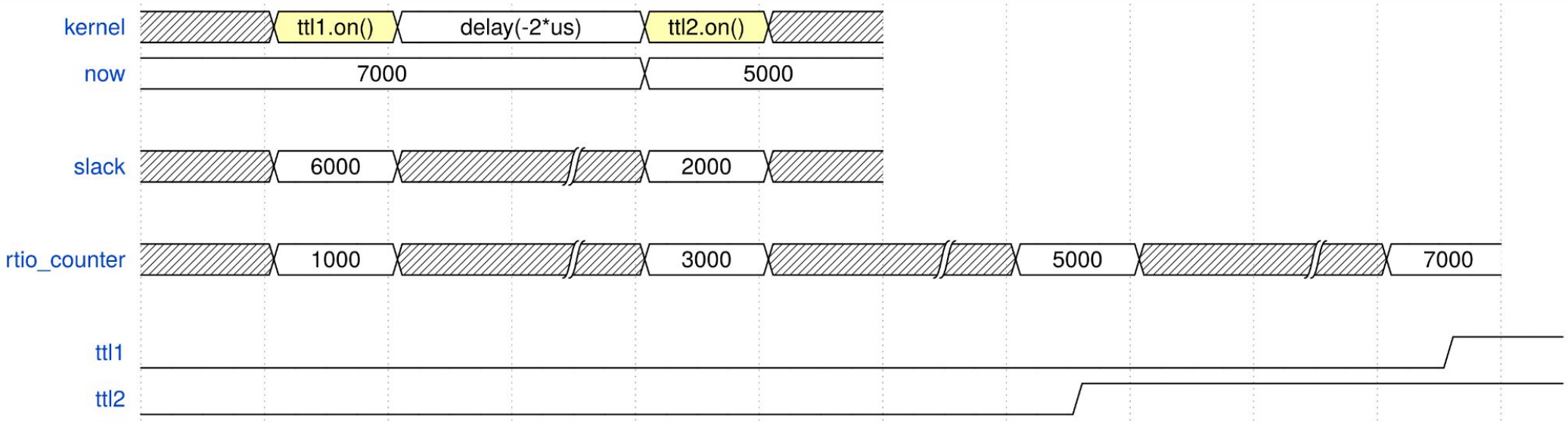


ARTIQ real-time concepts

Manipulating timeline

Delay:

```
delay(-2*us)
```



ARTIQ real-time concepts

Manipulating timeline

Current software counter (now):

```
somewhen_mu = now_mu()
```

Seconds to machine units:

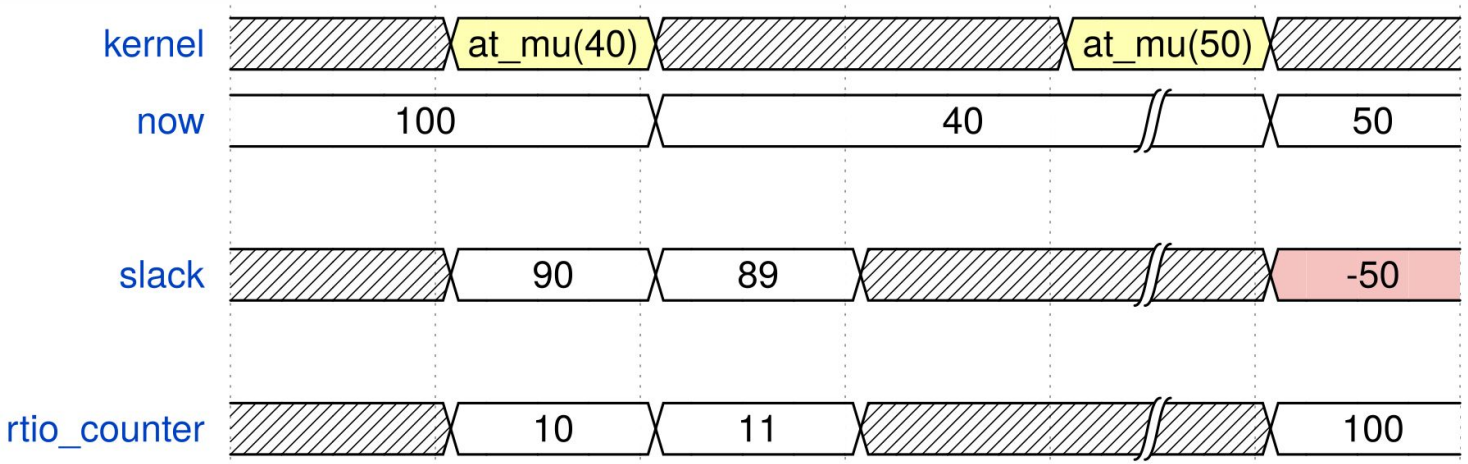
```
time_mu = \
  self.core.seconds_to_mu(12*us)
```

ARTIQ real-time concepts

Manipulating timeline

Setting software counter (*now* cursor):

```
at_mu(some_value_mu)
```

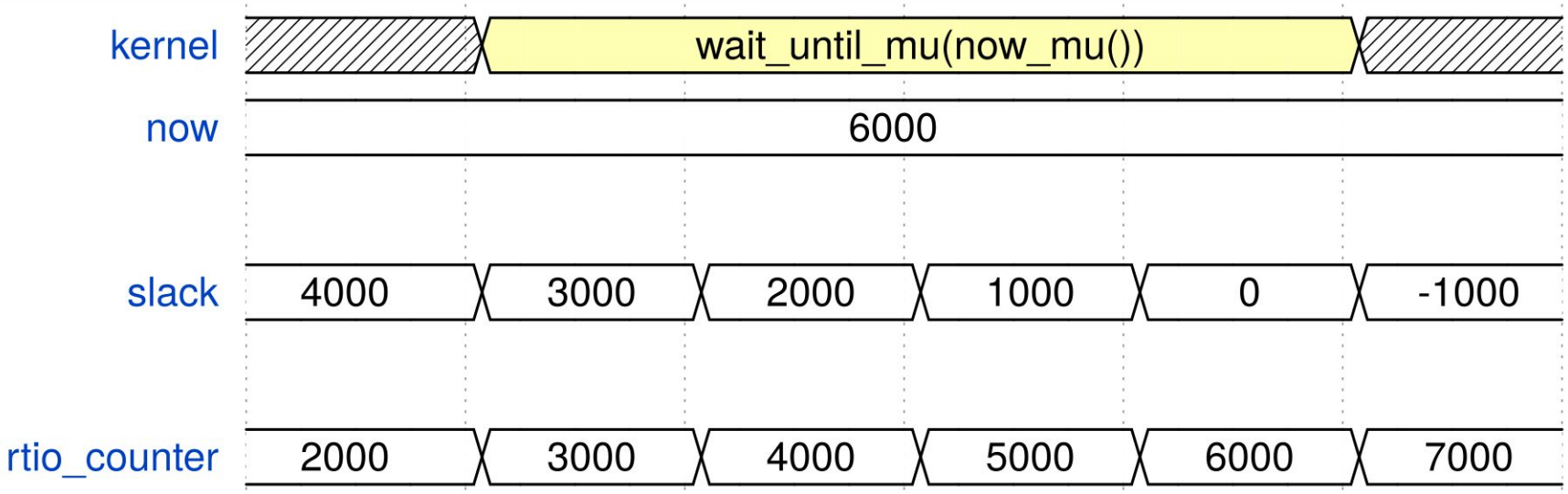


ARTIQ real-time concepts

Manipulating timeline

Hardware-software counter synchronization:

```
self.core.wait_unitl_mu(now_mu())
```

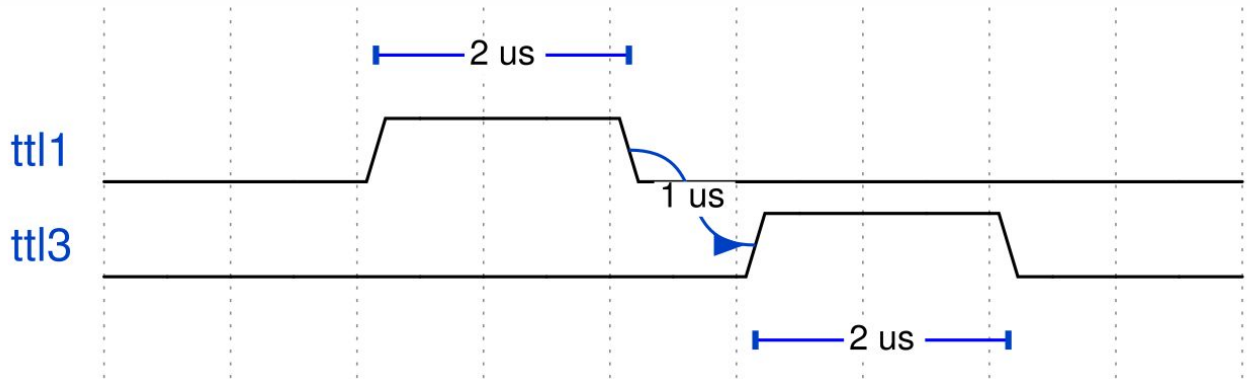


Timing examples - exercise Timing1

Exercise: `timing1.py`

Solution: `timing1_solution.py`

Goal: Create 2 pulses using two different TTL methods (on/off and pulse) and a **`delay`** function. Use two different TTL channels: **`self.ttl1`** and **`self.ttl3`**.



TTL methods:

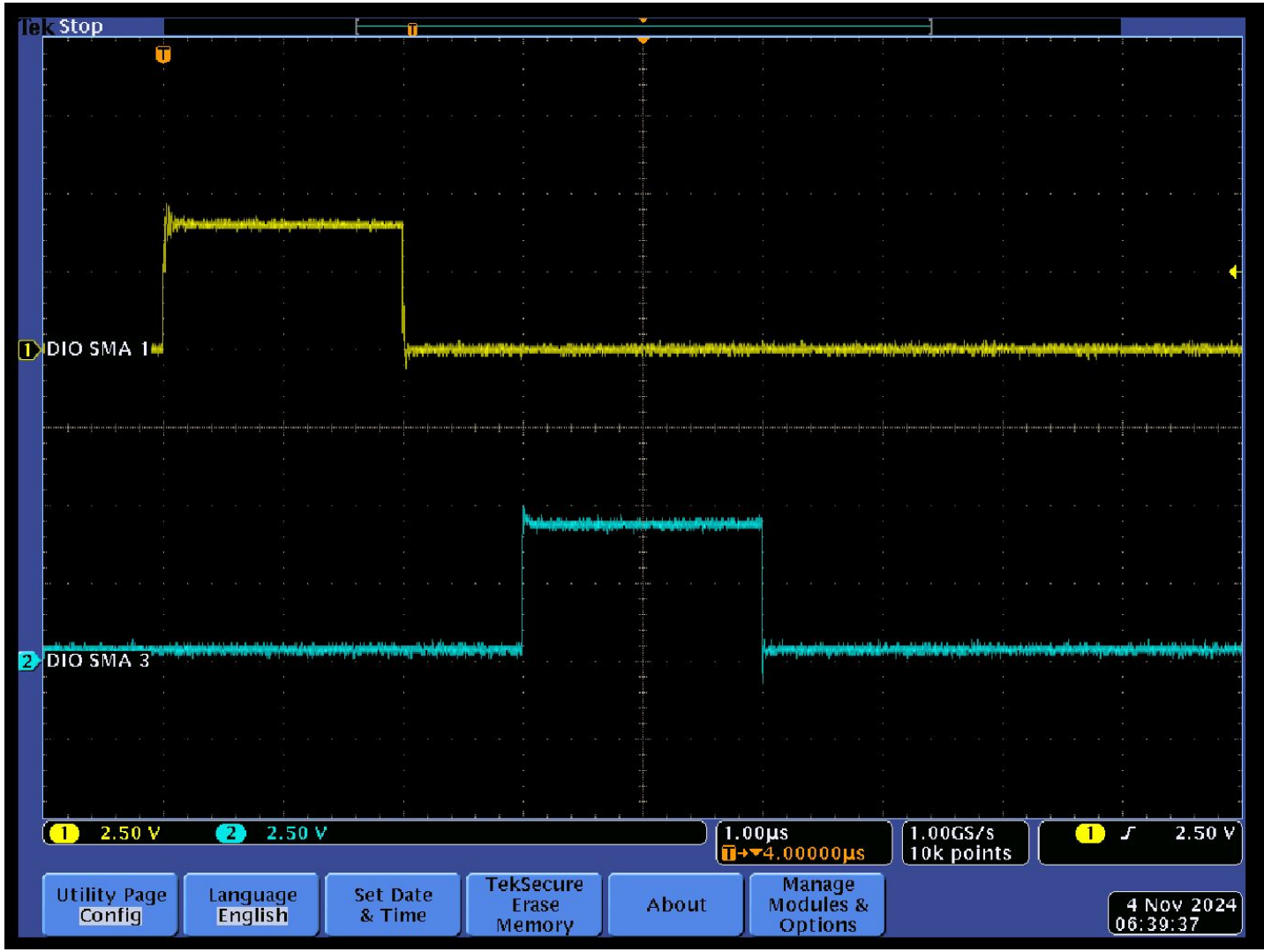
- `<TTL channel>.on()`
- `<TTL channel>.off()`
- `pulse(`*`duration`*`)` - **hidden delay inside!**

Timing functions:

- `delay(`*`duration`*`)`

Timing examples - exercise Timing1

```
self.ttl1.on()
delay(2*us)
self.ttl1.off()
delay(1*us)
self.ttl3.pulse(2*us)
```

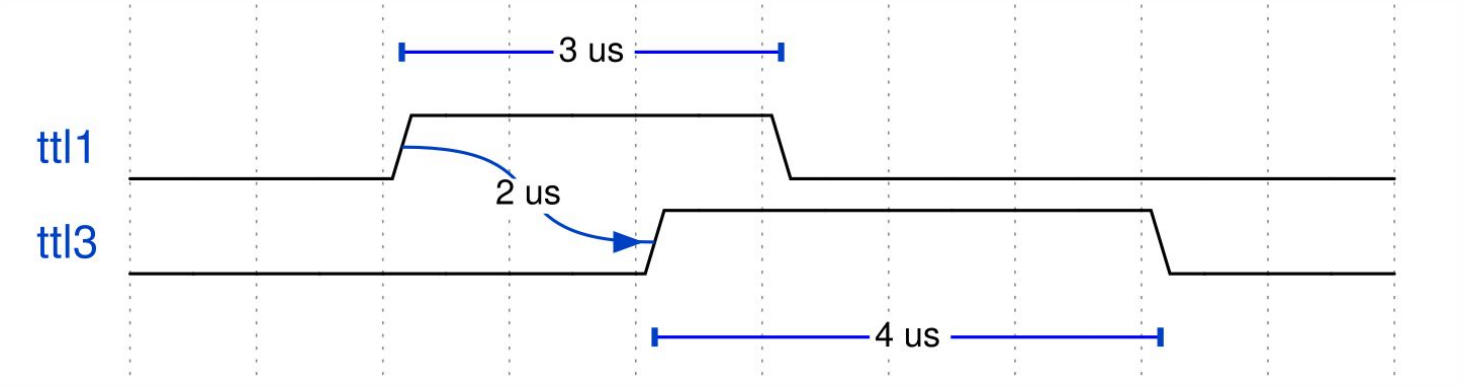


Timing examples - exercise Timing2

Exercise: `timing2.py`

Solution: `timing2_solution.py`

Goal: Create 2 pulses using two different TTL methods (on/off and pulse), **`now_mu()`** and **`at_mu()`** functions. Use two different TTL channels: **`self.ttl1`** and **`self.ttl3`**.



TTL methods:

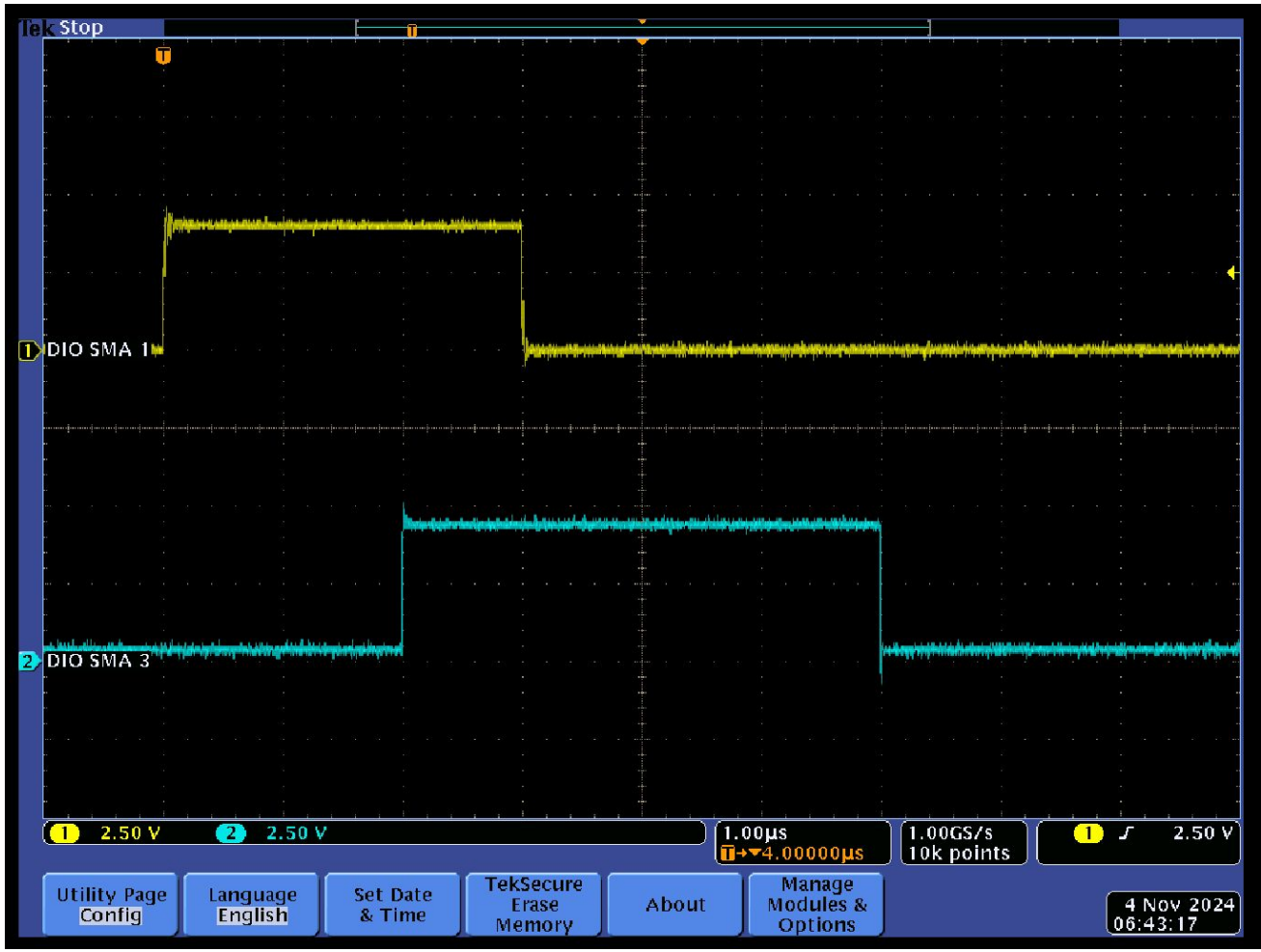
- `<TTL channel>.on()`
- `<TTL channel>.off()`
- `pulse(`*`duration`*`)` - **hidden delay inside!**

Timing functions:

- **`now_mu`**`(`*`duration`*`)`
- **`at_mu`**`(`*`time_mu`*`)`
- **`self.core.seconds_to_mu`**`(`*`time_in_sec`*`)`

Timing examples - exercise Timing2

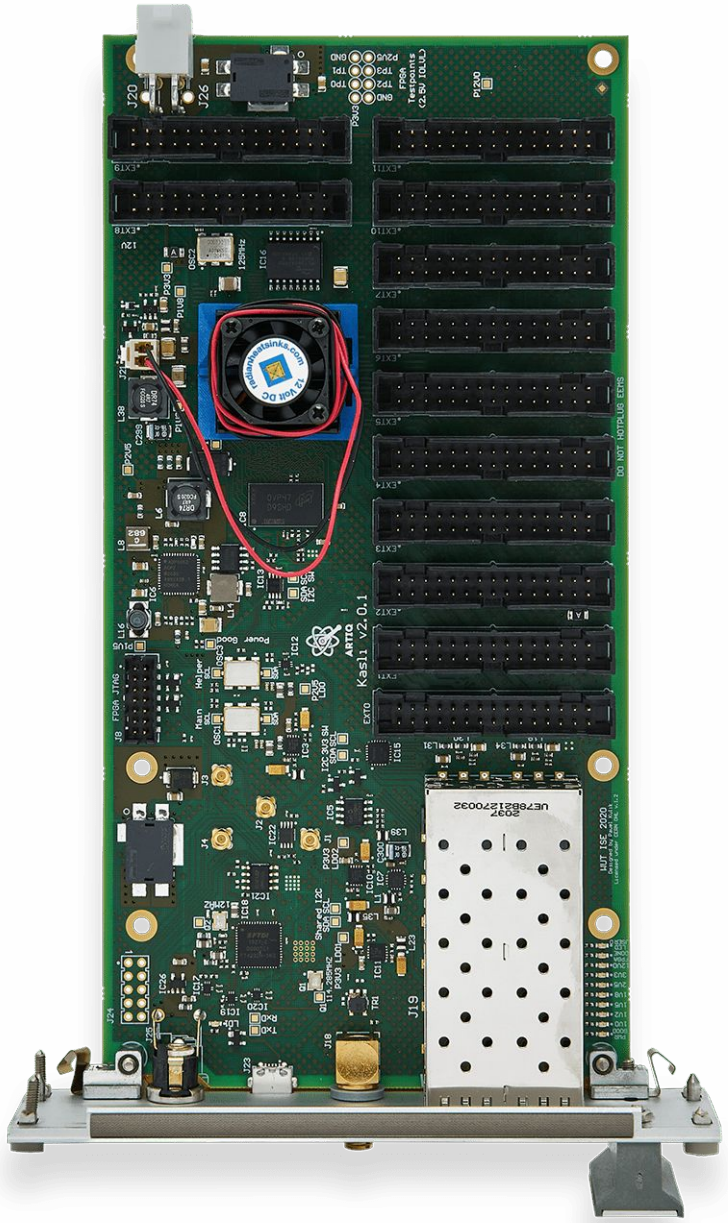
```
# We need to store the current counter
# value for later use
t = now_mu()

# This advances the counter by 3 us
self.ttl1.pulse(3*us)

# Let's move counter to the value
# corresponding to the start of the second
# pulse.
at_mu(t + self.core.seconds_to_mu(2*us))
self.ttl3.pulse(4*us)
```



ARTIQ DSL - what can be done in FPGA?

- Subset of Python:
  - Objects
  - Conditionals (if..else structure)
  - Loops, iterating over lists
  - Exceptions
- System specific:
  - Timing functions
  - Parallel / Sequential blocks
  - DMA
  - RPC

More detailed description: m-labs.hk/artiq/manual/compiler.html



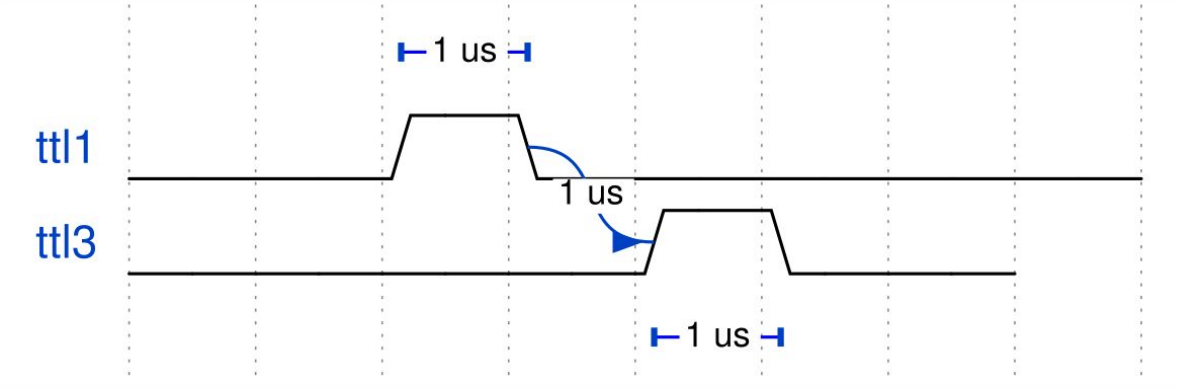
Warsaw University of Technology

Timing examples - exercise Timing3

Exercise: `timing3.py`

Solution: `timing3_solution.py`

Goal: Generate 1 us long pulse on **`self.ttl1`** followed by 1 us delay and 1 us long pulse on **`self.ttl3`**. Just after **`self.ttl1`** pulse print value of **`now_mu()`** using **`print()`** function.



TTL methods:

- `<TTL channel>.on()`
- `<TTL channel>.off()`
- `pulse(duration)` - **hidden delay inside!**

Timing and other functions:

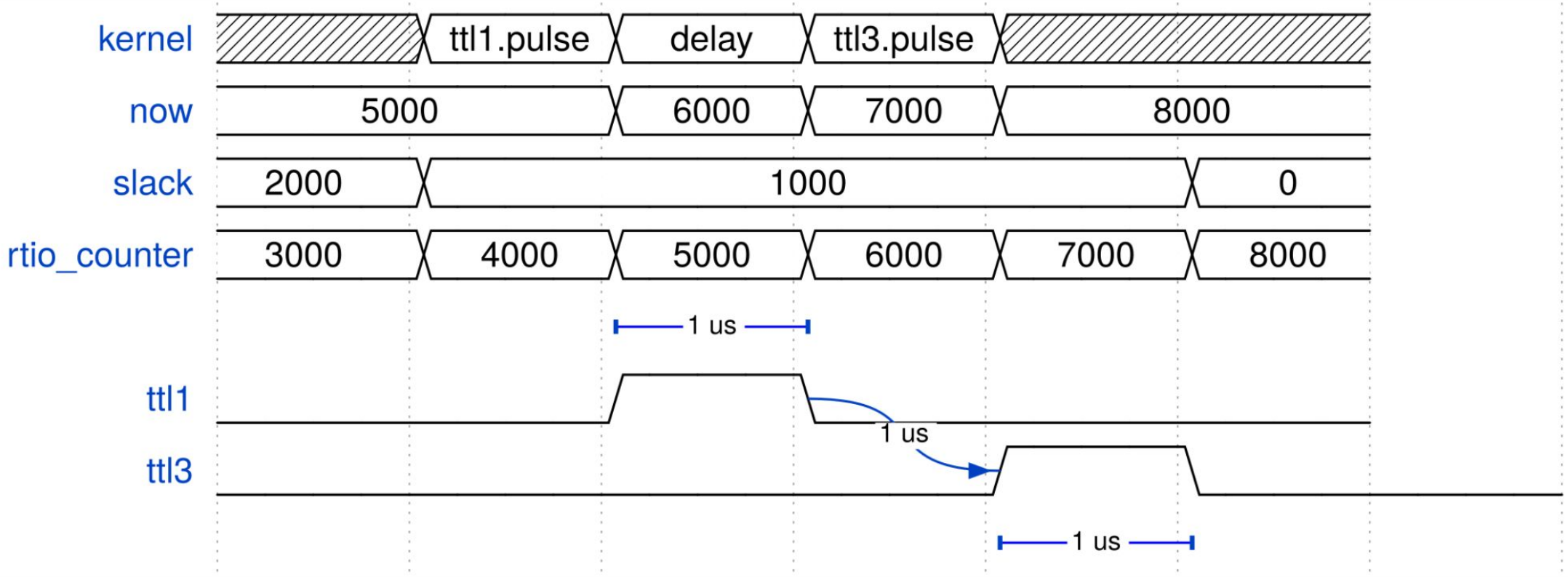
- **`now_mu`**`(duration)`
- **`print`**`(what)`

Warsaw University of Technology

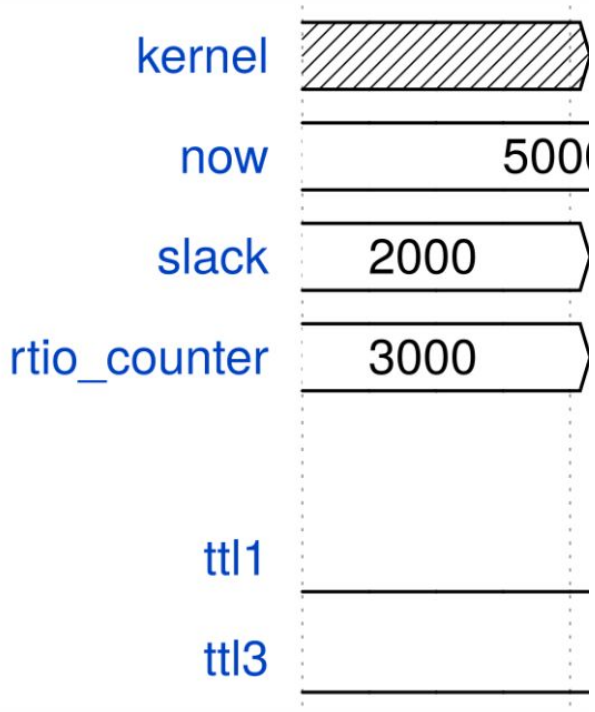
Timing examples - exercise Timing3

```
self.ttl1.pulse(1*us)
print(now_mu())
delay(1*us)
self.ttl3.pulse(1*us)
```

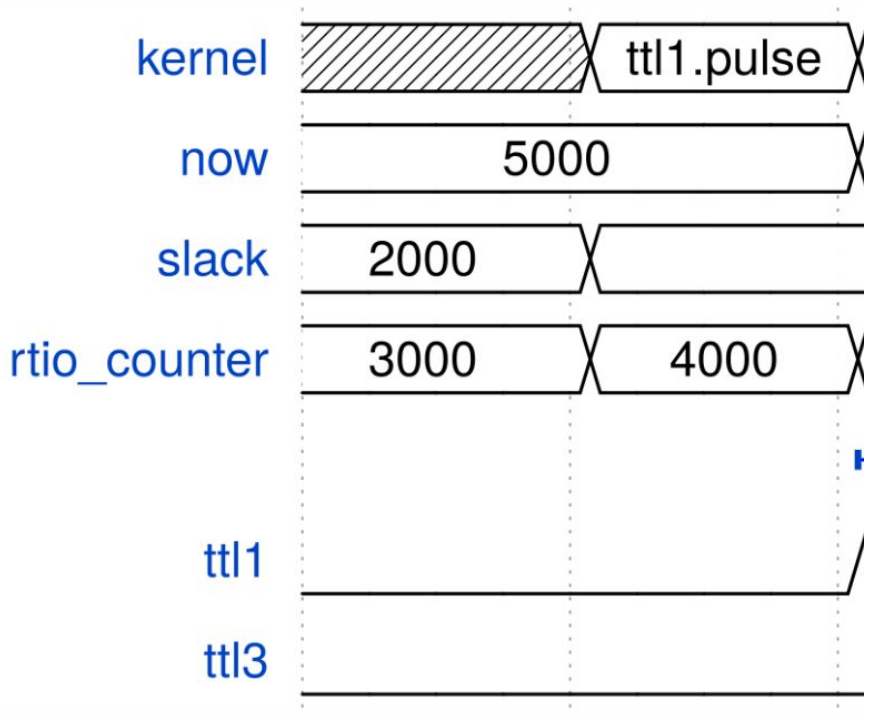
Timing examples - exercise Timing3 - why exception?



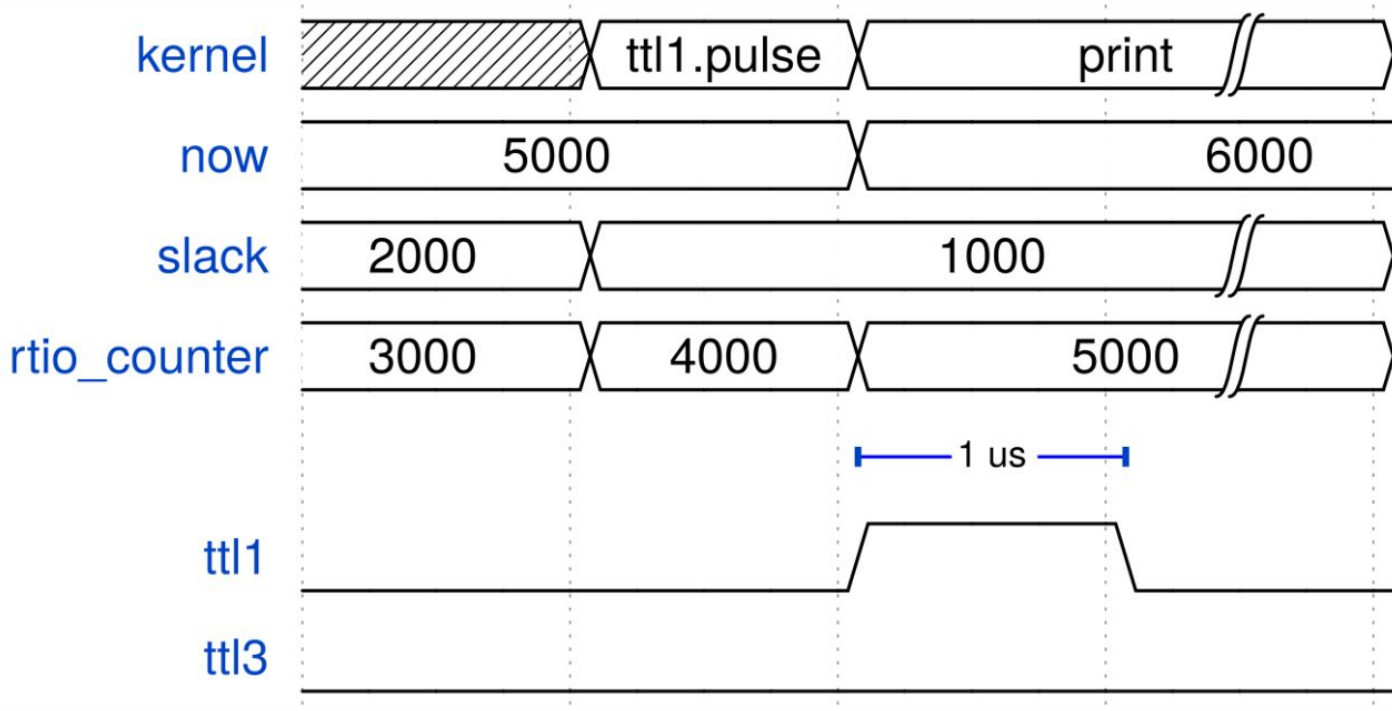
Timing examples - exercise Timing3 - why exception?



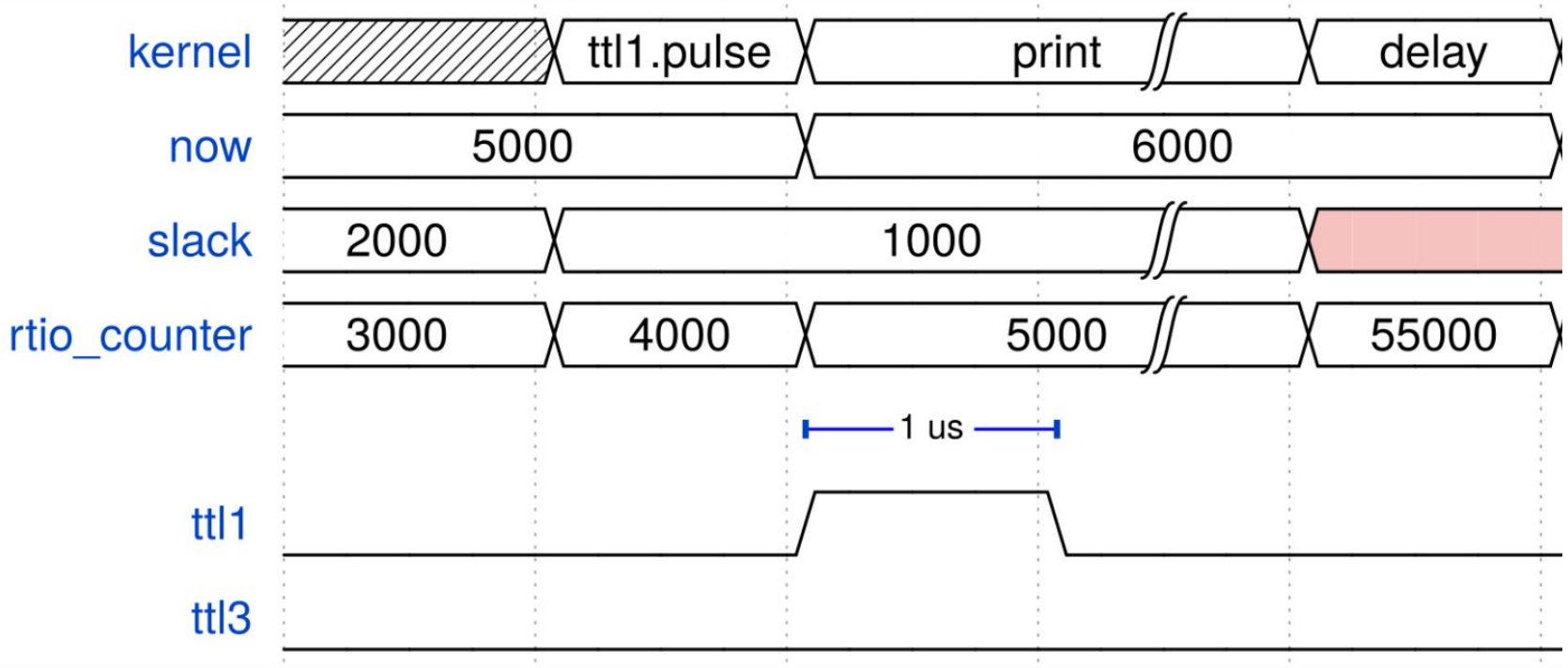
Timing examples - exercise Timing3 - why exception?



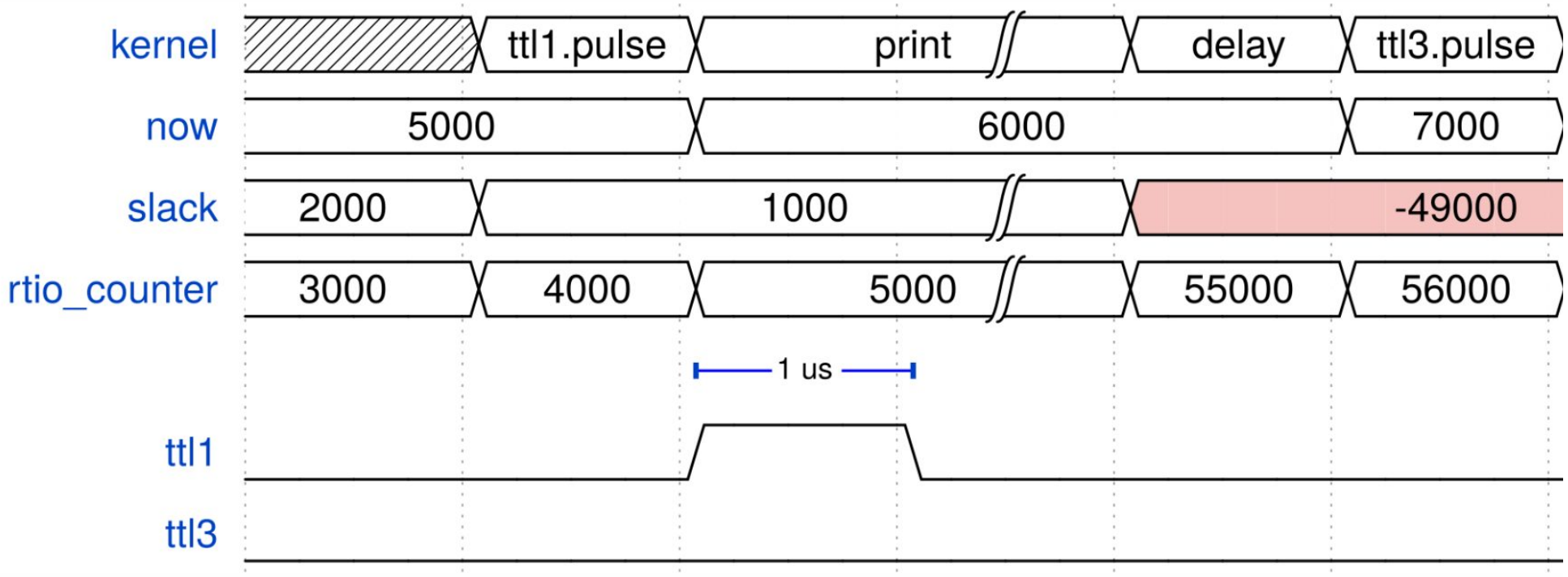
Timing examples - exercise Timing3 - why exception?



Timing examples - exercise Timing3 - why exception?



Timing examples - exercise Timing3 - why exception?

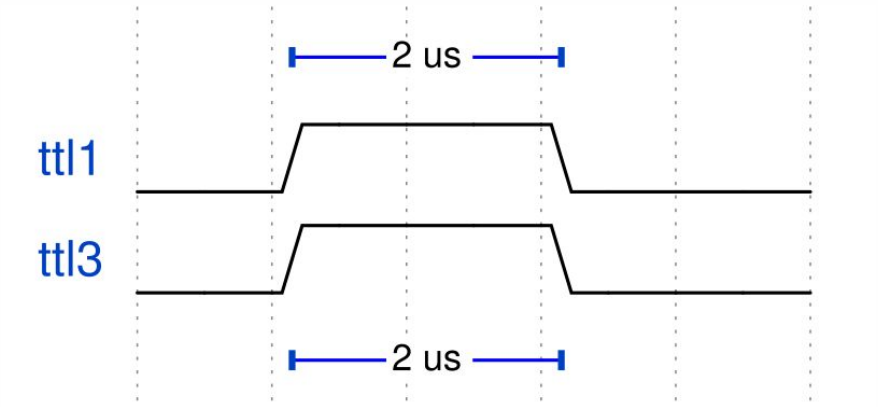


Altering the past

RTIOUnderflowException

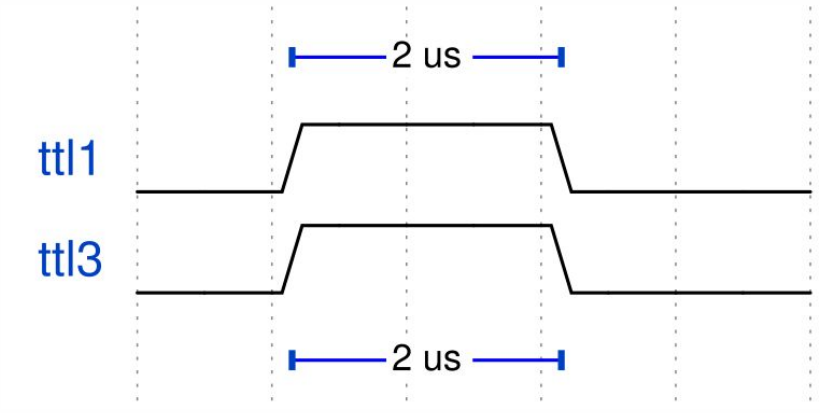
Parallel and sequential blocks

```
with parallel:
    self.ttl1.pulse(2*us)
    self.ttl3.pulse(2*us)
```

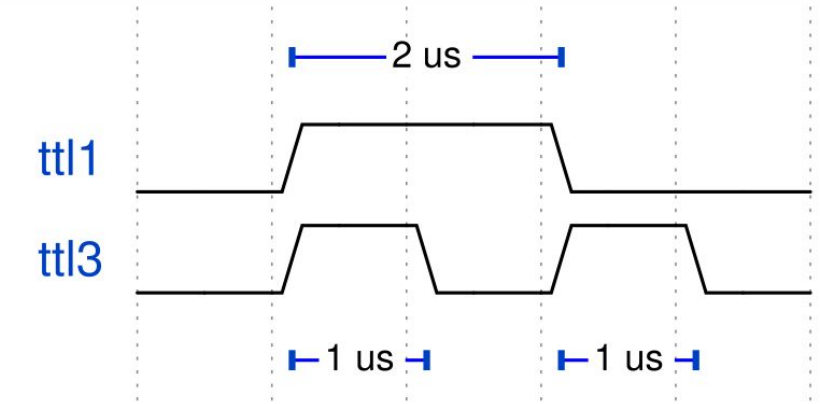


Parallel and sequential blocks

```
with parallel:
    self.ttl1.pulse(2*us)
    self.ttl3.pulse(2*us)
```

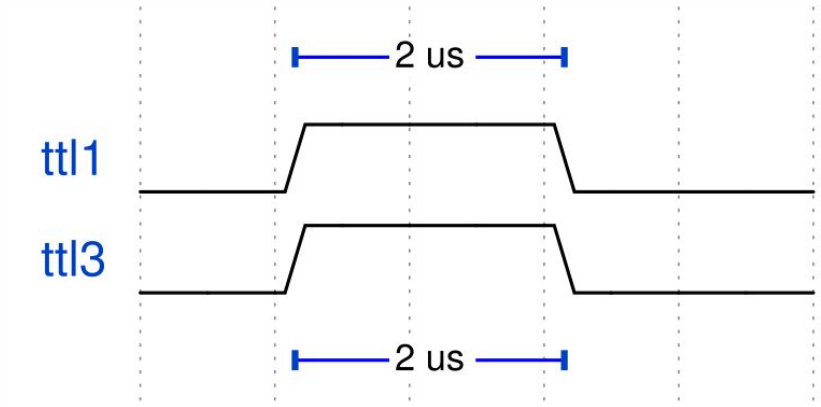



```
with parallel:
    self.ttl1.pulse(2*us)
    with sequential:
        self.ttl3.pulse(1*us)
        delay(1*us)
        self.ttl3.pulse(1*us)
```

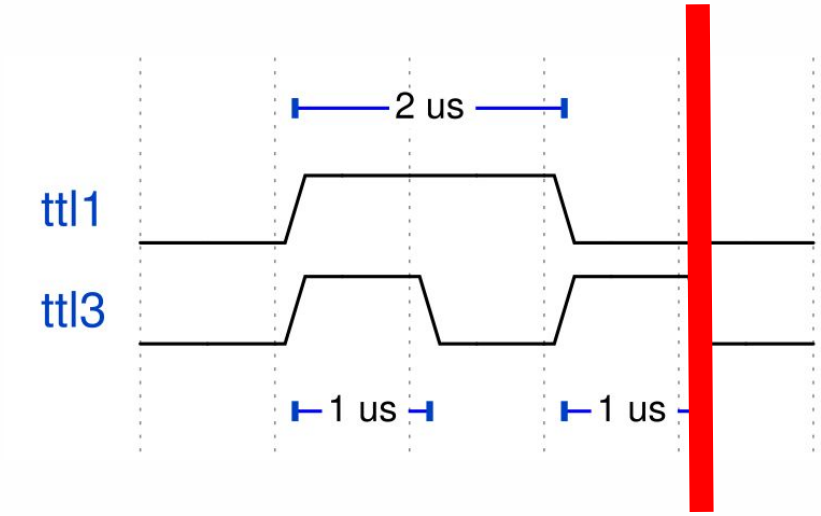


Parallel and sequential blocks

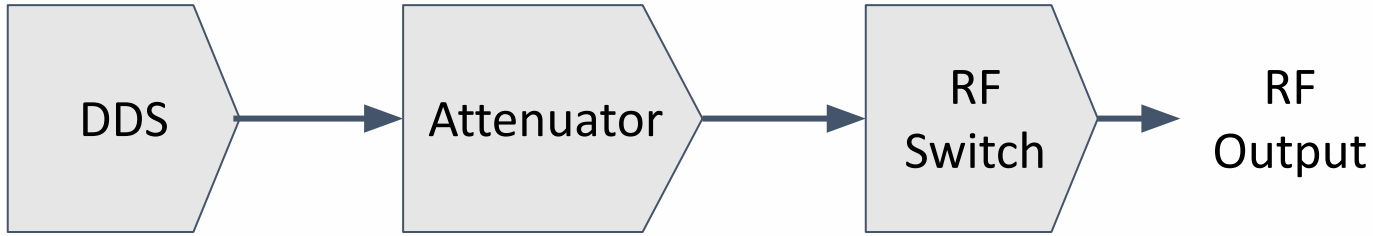
```
with parallel:
    self.ttl1.pulse(2*us)
    self.ttl3.pulse(2*us)
```




```
with parallel:
    self.ttl1.pulse(2*us)
    with sequential:
        self.ttl3.pulse(1*us)
        delay(1*us)
        self.ttl3.pulse(1*us)
```

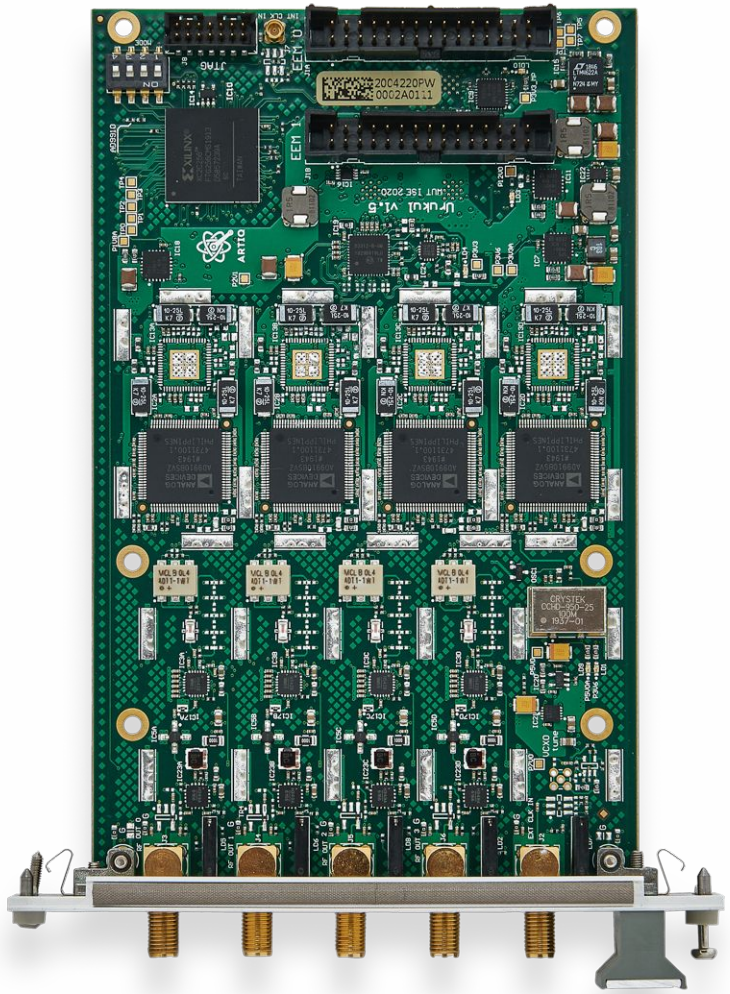


Controlling Urukul DDS



Urukul channel methods

- `channel.init()`
  initialize channel
- `channel.set(freq, phase, amplitude)`
  freq - float frequency in HZ
  phase - float phase tuning word in turns
  amplitude - float amplitude in units of full scale <0;1>
- `channel.set_att(att)`
  att - float attenuation in SI units [0 .. 31.5 dB]
- `channel.sw` - TTL controlling RF switch
  TTL output functions apply, i.e. on(), off(), pulse()



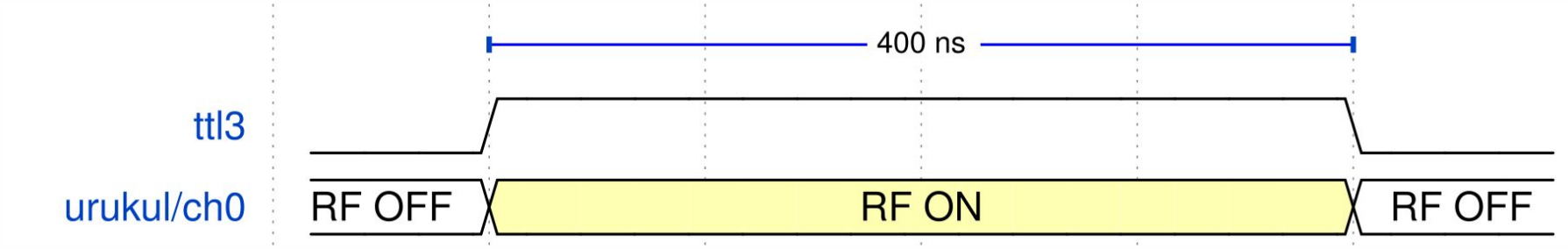
Warsaw University of Technology

TTL and Urukul - exercise TTLUrukul1

Exercise: `ttl_urukul1.py`

Solution: `ttl_urukul1_solution.py`

Goal: Generate simultaneous **`self.ttl3`** and Urukul channel 0 (**`self.urukul_channels[0].sw`**) pulse 400 ns long.



TTL methods:

- `<TTL channel>.on()`
- `<TTL channel>.off()`
- `pulse(duration)` - **hidden delay inside!**

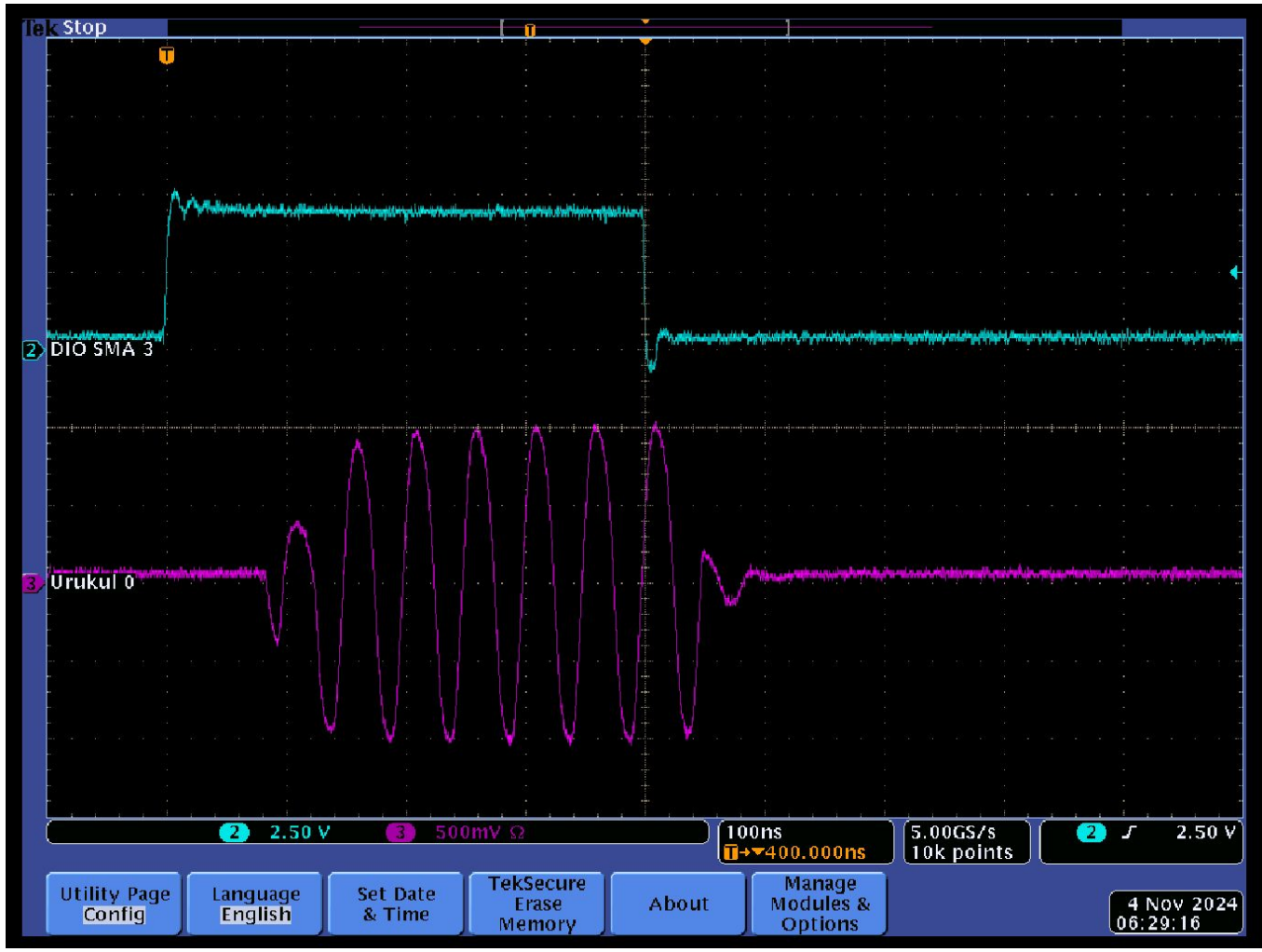
Urukul channel methods

- **`channel.sw`** - TTL controlling RF switch
  TTL output functions apply, i.e. on(), off(), pulse()

Warsaw University of Technology

TTL and Urukul - exercise TTLUrukul1

```
with parallel:
    self.ttl3.pulse(400 * ns)
    self.urukul_channels[0].sw.pulse(400 * ns)
```



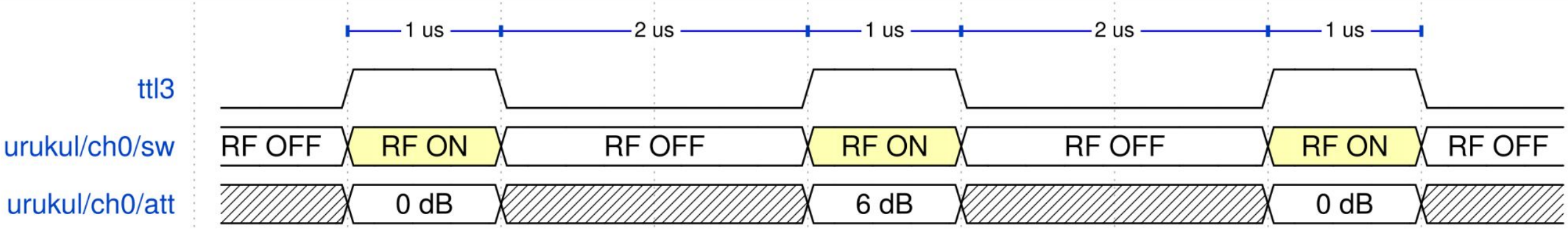
Warsaw University of Technology

TTL and Urukul - exercise TTLUrukul2

Exercise: `ttl_urukul2.py`

Solution: `ttl_urukul2_solution.py`

Goal: On **`self.ttl3`** generate 3 1 us pulses separated by 2 us delay. Enable Urukul ch. 0 RF output in parallel with TTL output, with the same pattern. Make first RF pulse have attenuation 0 dB, second 6 dB and final again 0 dB.



TTL methods:

- `<TTL channel>.on()`
- `<TTL channel>.off()`
- `pulse(duration)` - **hidden delay inside!**

Urukul channel methods

- **`channel.set_att(att_dB)`**
- **`channel.sw`** - TTL controlling RF switch
  TTL output functions apply, i.e. on(), off(), pulse()

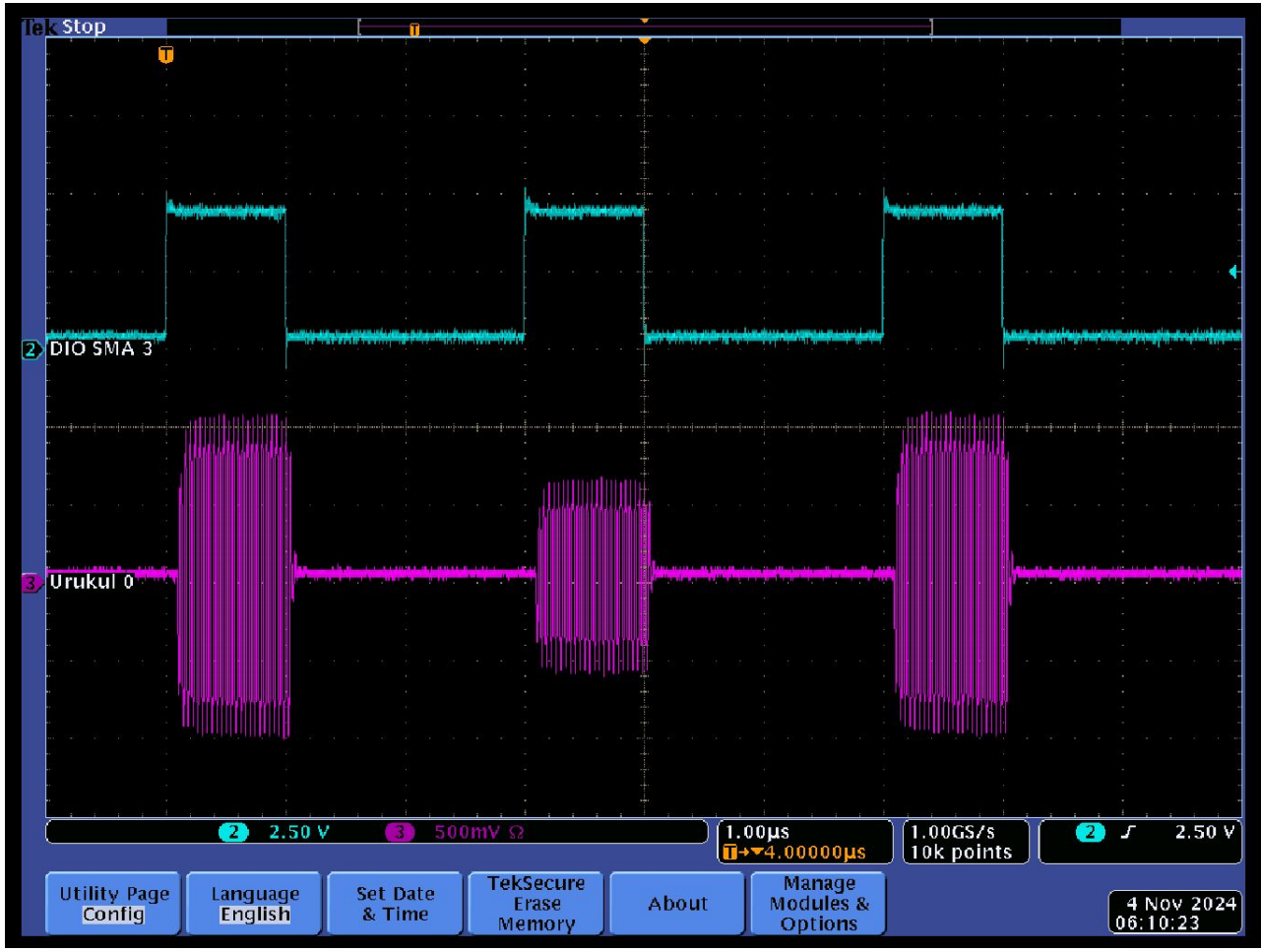
Timing functions:

- **`now_mu`**`(duration)`
- **`at_mu`**`(time_mu)`
- **`self.core.seconds_to_mu`**`(time_in_sec)`

Warsaw University of Technology

TTL and Urukul - exercise TTLUrukul2

```
t = now_mu()
with parallel:
      with sequential:
            for _ in range(3):
                        self.ttl3.pulse(1*us)
                        delay(2*us)
      with sequential:
            # t + 0 us
            self.urukul_channels[0].sw.pulse(1*us)
            self.urukul_channels[0].set_att(6.0)
            # t + 3 us
            at_mu(t + self.core.seconds_to_mu(3*us))
            self.urukul_channels[0].sw.pulse(1*us)
            self.urukul_channels[0].set_att(0.0)
            # t + 6 us
            at_mu(t + self.core.seconds_to_mu(6*us))
            self.urukul_channels[0].sw.pulse(1*us)
```



Warsaw University of Technology

DIO - a closer look at the TTL module

Connections:

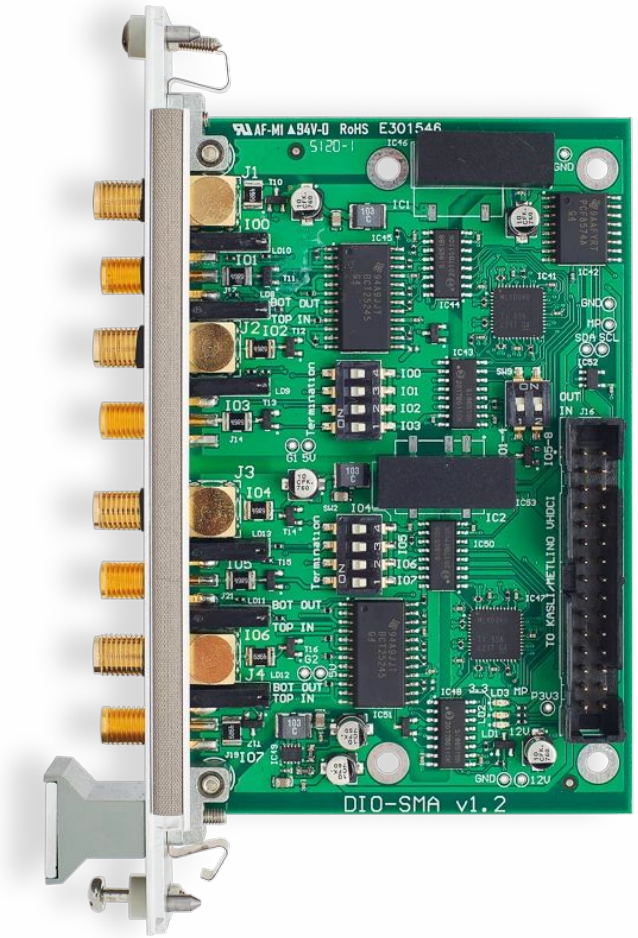
- TTL channel 1 - output to scope CH1 and TTL channel 5
- TTL channel 3 - output to scope CH2
- TTL channel 5 - input with signal fed from channel 1

Common TTL methods:

- `on()`
- `off()`
- `pulse(`*`duration`*`)` - **hidden delay inside!**
- `sample_input()`
- `get_sample()`
- `gate_rising(`*`duration_sec`*`)/gate_falling(`*`duration_sec`*`)/gate_both(`*`duration_sec`*`)`
- `count()`
- `timestamp_mu(`*`timestam_mu`*`)`

Optional:

- controller may be equipped with with gateware edge counter



Warsaw University of Technology

TTL sampling input methods

TTL channel methods:

- `sample_input()` - instructs the system to sample input at current **now** time marker position
- `sample_get()` - returns value previously sampled at **current wall clock position**

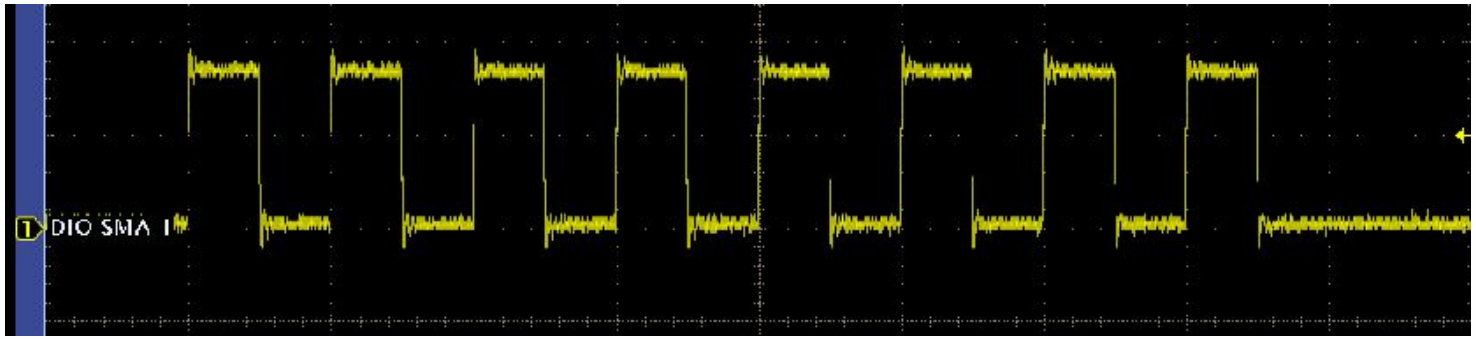
```
for _ in range(5):
    self.ttl5.sample_input()
    delay(1 * us)

for _ in range(5):
    self.ttl5.sample_get()
```

TTL examples - exercise TTLSample

Exercise: `ttl_sample.py`

Solution: `ttl_sample_solution.py`

Goal: There is a square wave signal generated on TTL channel **`self.ttl1`** that lasts for 8 us and has period of 1us. Sample signal fed to **`self.ttl5`** exactly in the middle of each state, put these values inside **levels** list and print it. You should be able to see a list of the following contents: [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0].



TTL methods:

- `<TTL channel>.sample_input()`
- `<TTL channel>.sample_get()`

Timing functions:

- **`now_mu`**`(`*`duration`*`)`
- **`at_mu`**`(`*`time_mu`*`)`
- **`self.core.wait_until_mu`**`(`*`time_in_sec`*`)`

TTL gate window methods

TTL channel methods:

- `gate_rising(duration_in_seconds)` - **hidden delay!**
- `gate_falling(duration_in_seconds)` **-hidden delay!**
- `gate_both(duration_in_seconds)` - **hidden delay!**
- `count(time_mu)`

```
t0 = now_mu()
gate_end_mu = self.ttl5.gate_falling(100 * ns)

t1 = now_mu()
received = self.ttl5.count(gate_end_mu)

t2 = now_mu()
```

```
t1 = t0 + 100 * ns
t2 = t1
```

Warsaw University of Technology

TTL examples - exercise TTLGatedInput

Exercise: `ttl_gated_input.py`

Solution: `ttl_gated_input_solution.py`

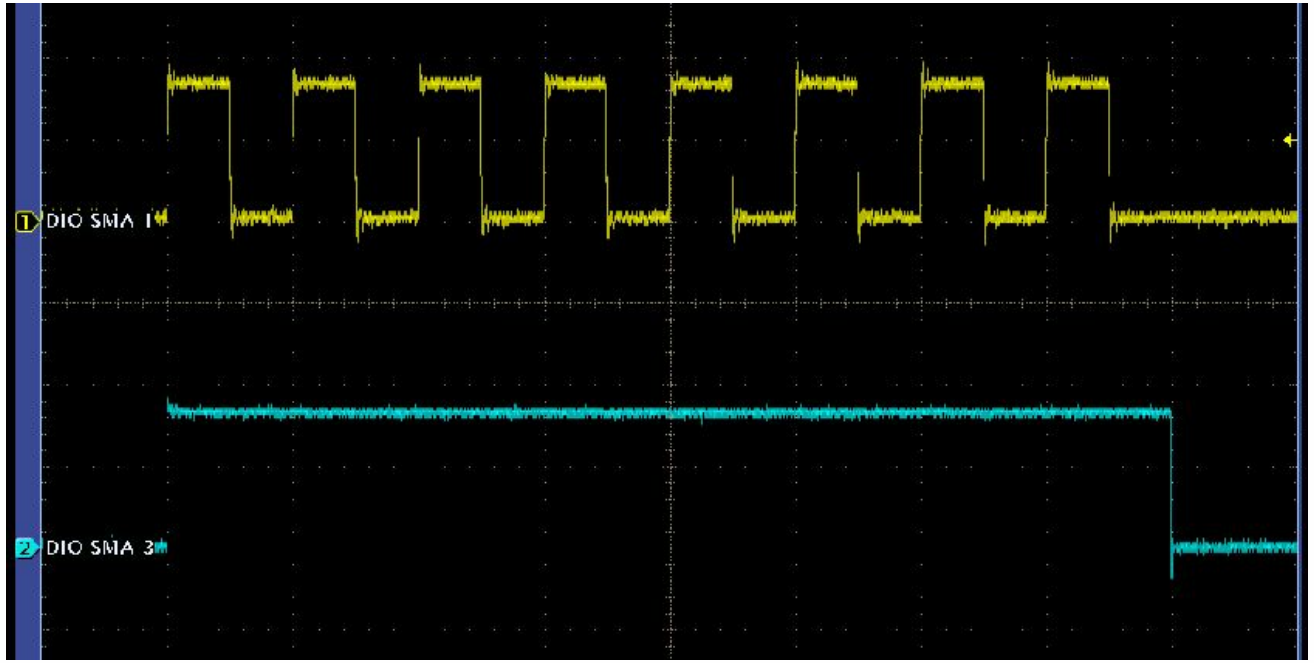
Goal: There is a square wave signal generated on TTL channel **`self.ttl1`** that lasts for 8 us and has period of 1us. Count both rising and falling edges of signal fed from **`self.ttl1`** to **`self.ttl5`**. Use **`self.ttl3`** output channel as an indicator of when the gate is open. Print the the number of rising and falling edges - expect 16 of them.

Timing functions:

- **`now_mu`**`(`*`duration`*`)`
- **`at_mu`**`(`*`time_mu`*`)`
- **`self.core.wait_until_mu`**`(`*`time_in_sec`*`)`

TTL methods:

- `<TTL channel>.gate_both(`*`time_in_sec`*`)`
- `<TTL channel>.count(`*`time_in_mu`*`)`
- `<TTL channel>.on()`
- `<TTL channel>.off()`
- `<TTL channel>.pulse(`*`duration`*`)`



Warsaw University of Technology

TTL examples - exercise TTLGatedInput cont.

Exercise: `ttl_gated_input.py`

Solution: `ttl_gated_input_solution.py`

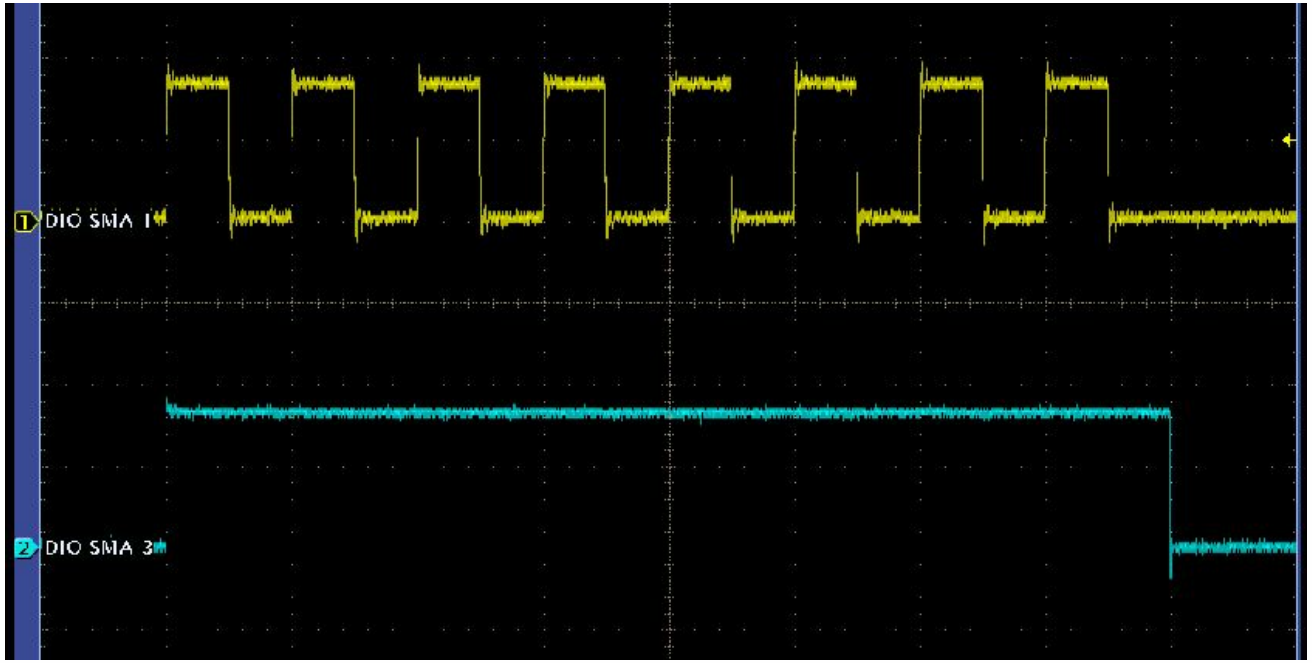
Goal: Find out when RTIOOverflow exceptions occurs.
Hint: change the number of events that are to be recorded within the gating period

Timing functions:

- **`now_mu`**`(`*`duration`*`)`
- **`at_mu`**`(`*`time_mu`*`)`
- **`self.core.wait_until_mu`**`(`*`time_in_sec`*`)`

TTL methods:

- `<TTL channel>.gate_both(`*`time_in_sec`*`)`
- `<TTL channel>.count(`*`time_in_mu`*`)`



Warsaw University of Technology

TTL gate window methods, cont.

TTL channel methods:

- gate_rising(*duration_in_seconds*) - **hidden delay!**
- gate_falling(*duration_in_seconds*) -**hidden delay!**
- gate_both(*duration_in_seconds*) - **hidden delay!**
- **timestamp_mu**(*time_mu*)

timestamp_mu(*time_mu*):

- returns a **timestamp** of the next event gathered in input FIFO, or
- -1 if no event has been recorded within the duration of gate window

```
gate_end_mu = self.ttl5.gate_falling(100 * ns)

result = self.ttl5.timestamp_mu(gate_end_mu)
```

TTL examples - exercise TTLGatedTimestamp

Exercise: `ttl_gated_timestamp.py`

Solution: `ttl_gated_timestamp_solution.py`

Goal: There is a square wave signal generated on TTL channel **`self.ttl1`** that lasts for 8 us and has period of 1us. Open gate window for both rising and falling edges of signal fed from **`self.ttl1`** to **`self.ttl5`**, but instead of counting them, retrieve each event's timestamp and insert into **`self.timestamps`** list. You should be able to see these events' timestamps in relation to **`self.t0`** printed out.
Note: **self.timestamps** can hold only 8 elements, so you must make sure not to exceed it's indexes.

```
. print:timestamp-t0 [us]:  0.16 0.66 1.16 1.66 2.16 2.66 3.16 3.66
```

TTL methods:

- `<TTL channel>.gate_both(time_in_sec)`
- `<TTL channel>.timestamp_mu(time_in_mu)`

TTL examples - exercise TTLEdgeCounter

Exercise: `ttl_edge_counter.py`

Solution: `ttl_edge_counter_solution.py`

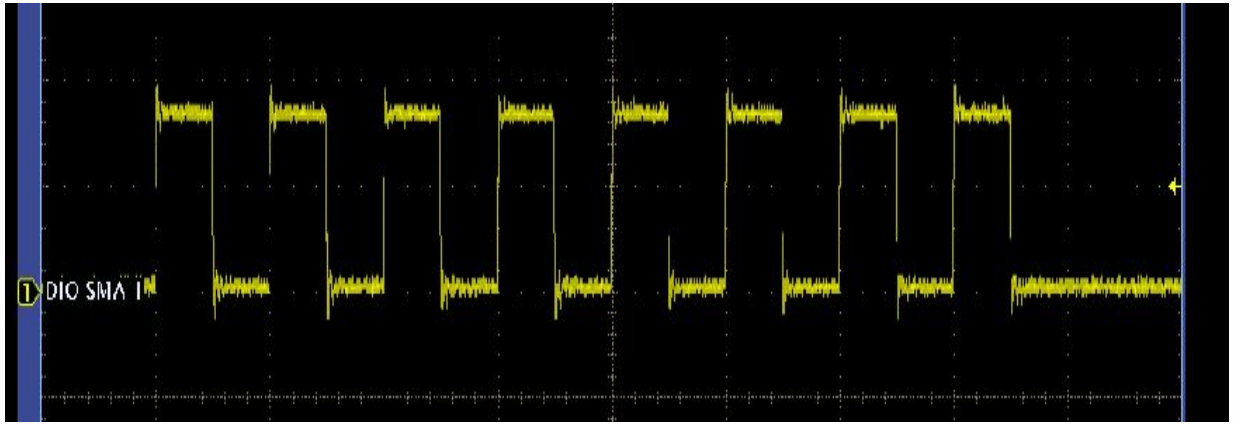
Goal: Write missing part of the experiment that generates square-like signal **`self.ttl1`** that lasts 60 us and has period of 1 us. This signal is fed by wire to **`self.ttl5`**. Count both rising and falling edges of using **`self.ttl_edge_counter`** device. Print the number of rising and falling edges. You should see 120 of them. NOTE: use ***PERIOD_US*** and ***N_PULSES*** variables to calculate gate duration and to drive outputs with ***PERIOD_US*** period.

Useful functions:

- **`parallel`** `blocks`
- **`sequential`** `blocks`

TTL methods:

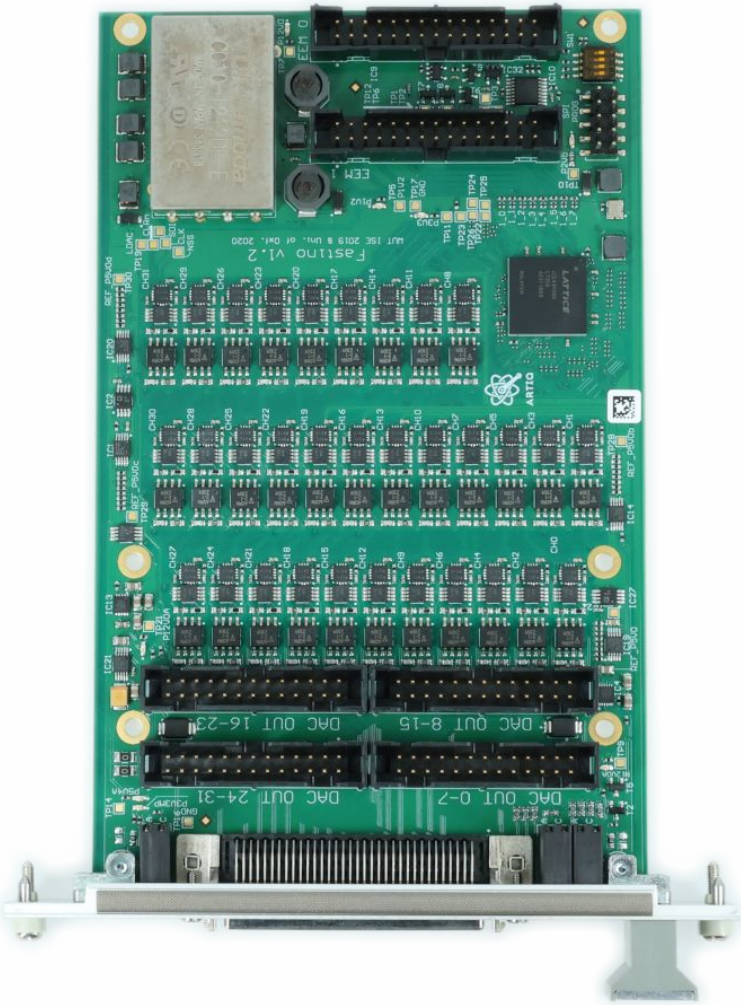
- `<TTL channel>.pulse(`*`duration_in_sec`*`)`
- `<TTL EdgeCounter>.gate_both(`*`time_in_mu`*`)`
- `<TTL EdgeCounter>.fetch_count()`



Warsaw University of Technology

Devices that will be used

- Fastino
  - 32 channel, 16-bit, 2.55 MSPS DAC
  - + / - 10 V output range
  - 1 us settlings time
  - https://github.com/sinara-hw/Fastino/wiki

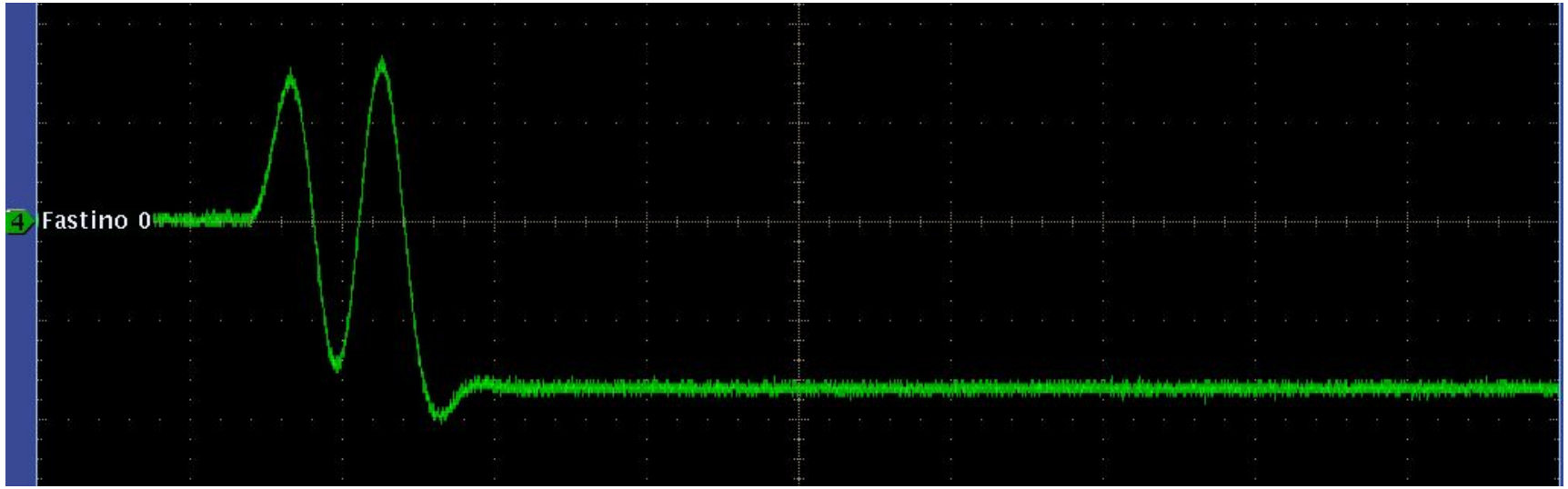


Basic Fastino (DAC) output - exercise FastinoBasic

Exercise: `fastino_basic.py`

Solution: `fastino_basic_solution.py`

Goal: Output any sine wave on Fastino channel. Generate samples in a provided loop. Parametrize your code using **Amplitude** and **sample_num**. Then try to increase **sample_num** and delay multiplier.



Fastino methods:

- **`set_dac`**`(dac=0, voltage)`

Timing functions:

- **`delay`**`(`*`duration`*`)`

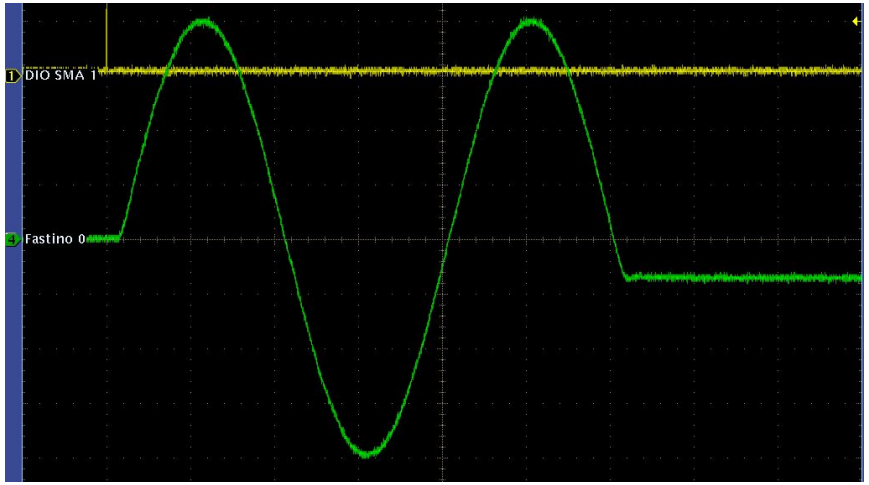
Warsaw University of Technology

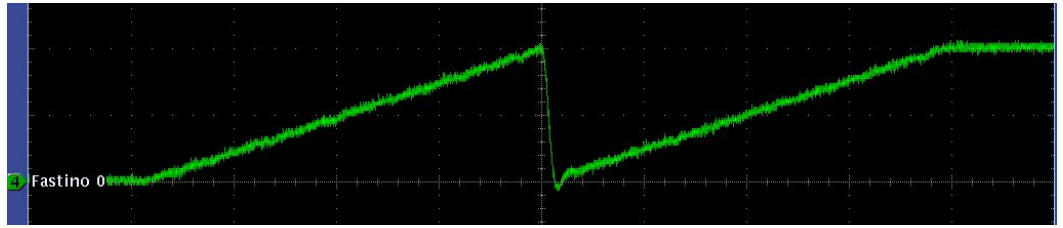
Precalculating waveforms - exercise FastinoPrepare

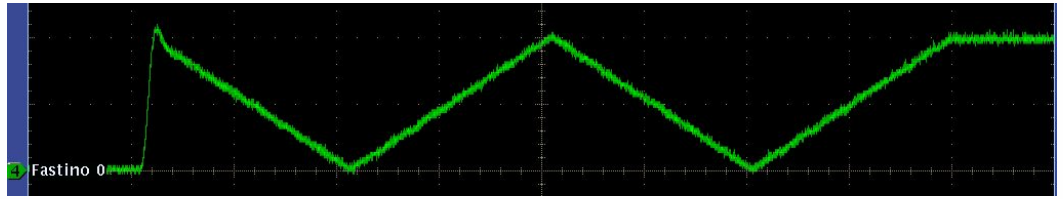
Exercise: `fastino_prepare.py`

Solution: `fastino_prepare_solution.py`

Goal: Write a loop that calculates a sine wave in a **prepare** function. Parametrize your code using **Amplitude** and **sample_num**. Then write code that prepares samples with square, sawtooth and triangle functions. Normalize your sequence to **Amplitude.** Then try changing **sample_num** and delay multiplier. What is the maximum number of samples now?







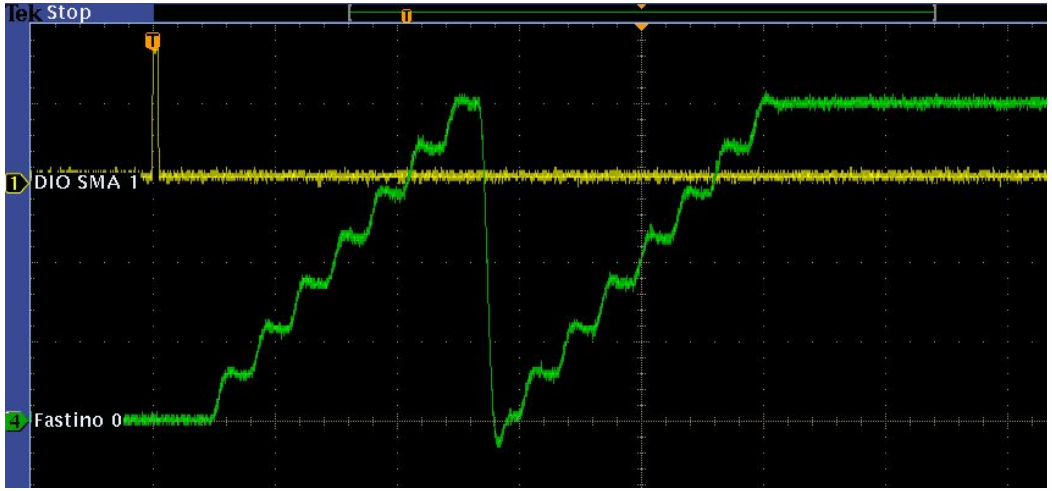
Methods:

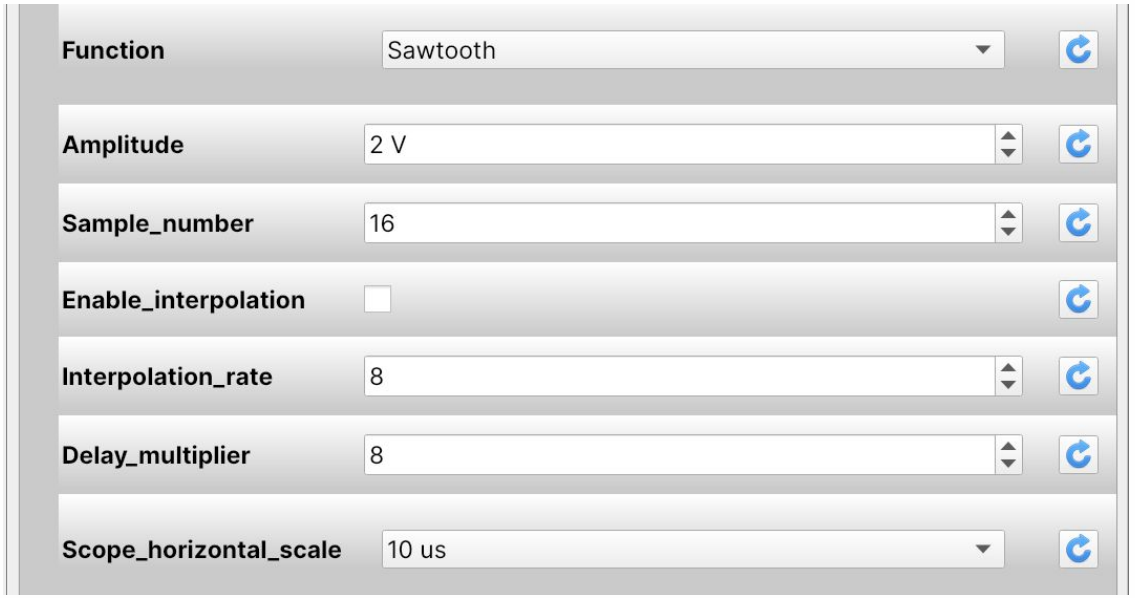
- List comprehension
- Change oscilloscope horizontal scale in dashboard when needed

Interpolation - exercise FastinoInterpolation

Exercise: `fastino_interpolation.py`

Goal: Use dashboard to change parameters, functions, enable/disable interpolation, see how it changes output shape, levels, delay.





Notes:

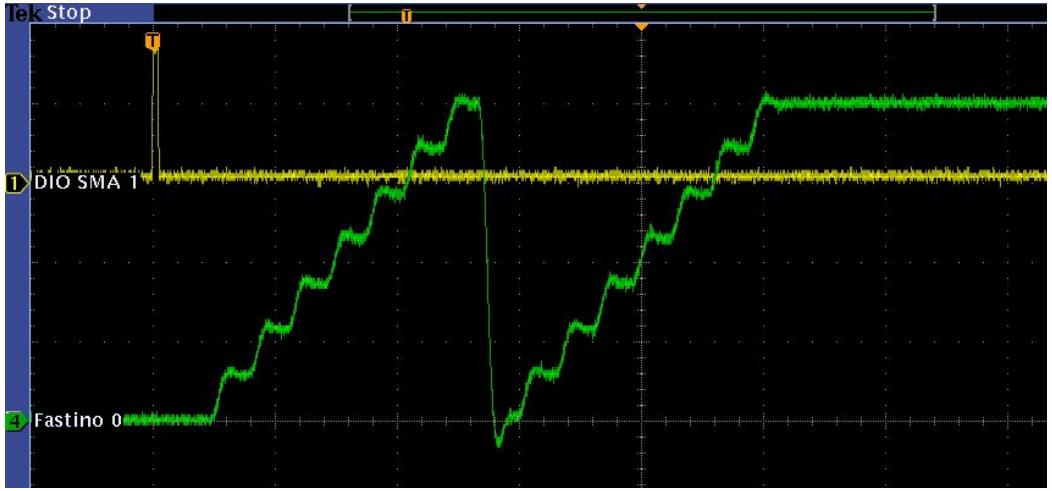
- You may copy your functions from previous exercise to fastino_interpolation.py.
- When interpolation is enabled Fastino will only accept one input sample per input sample period.

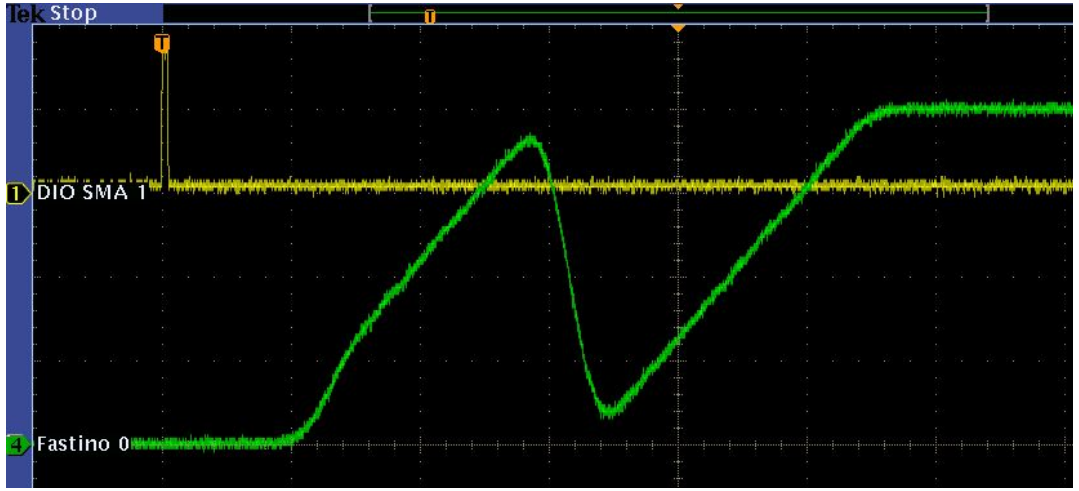
Warsaw University of Technology

Interpolation - exercise FastinoInterpolation

Exercise: `fastino_interpolation.py`

Goal: Use dashboard to change parameters, functions, enable/disable interpolation, see how it changes output shape, levels, delay.





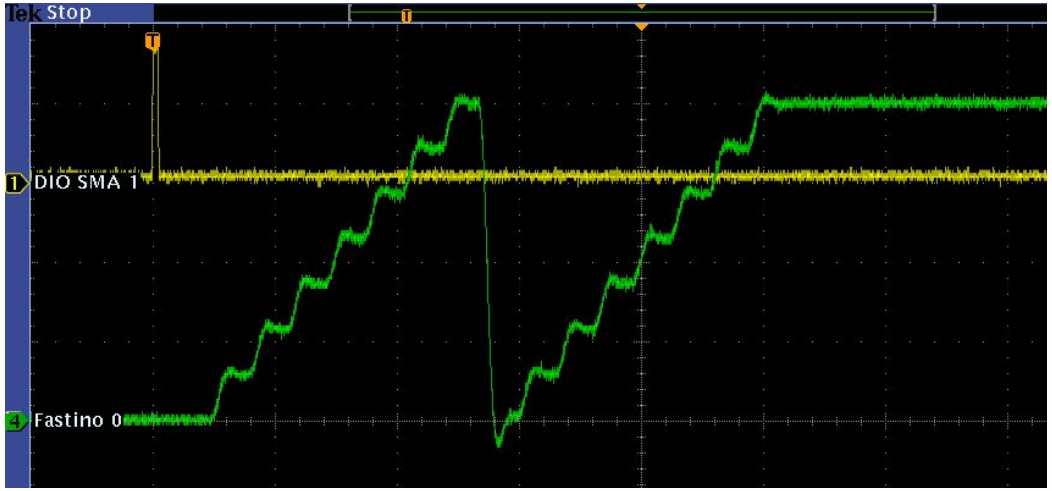
Notes:

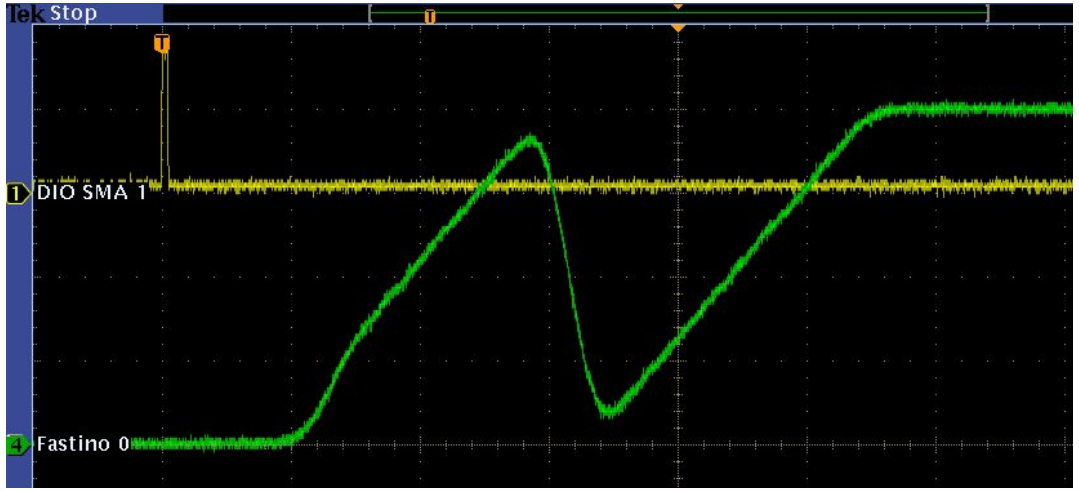
- You may copy your functions from previous exercise to fastino_interpolation.py.
- When interpolation is enabled Fastino will only accept one input sample per input sample period.

Interpolation - exercise FastinoInterpolation

Exercise: `fastino_interpolation.py`

Goal: Use dashboard to change parameters, functions, enable/disable interpolation, see how it changes output shape, levels, delay.



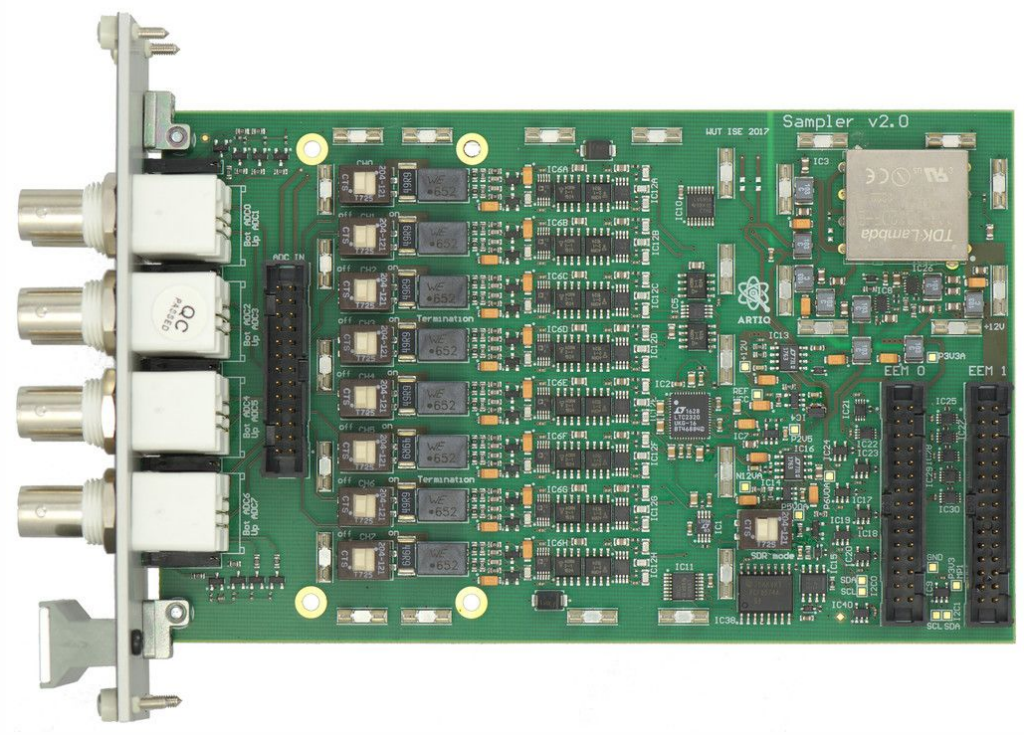


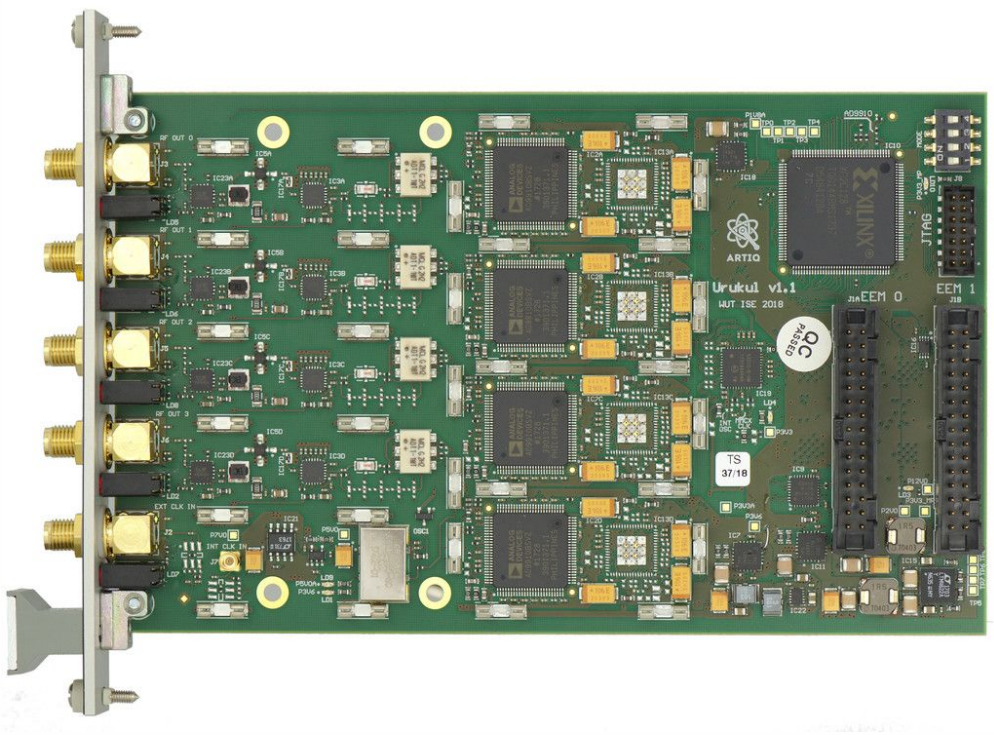
Notes:

- You may copy your functions from previous exercise to fastino_interpolation.py.
- When interpolation is enabled Fastino will only accept one input sample per input sample period.

Use cases

Sampler ADC + Urukul DDS





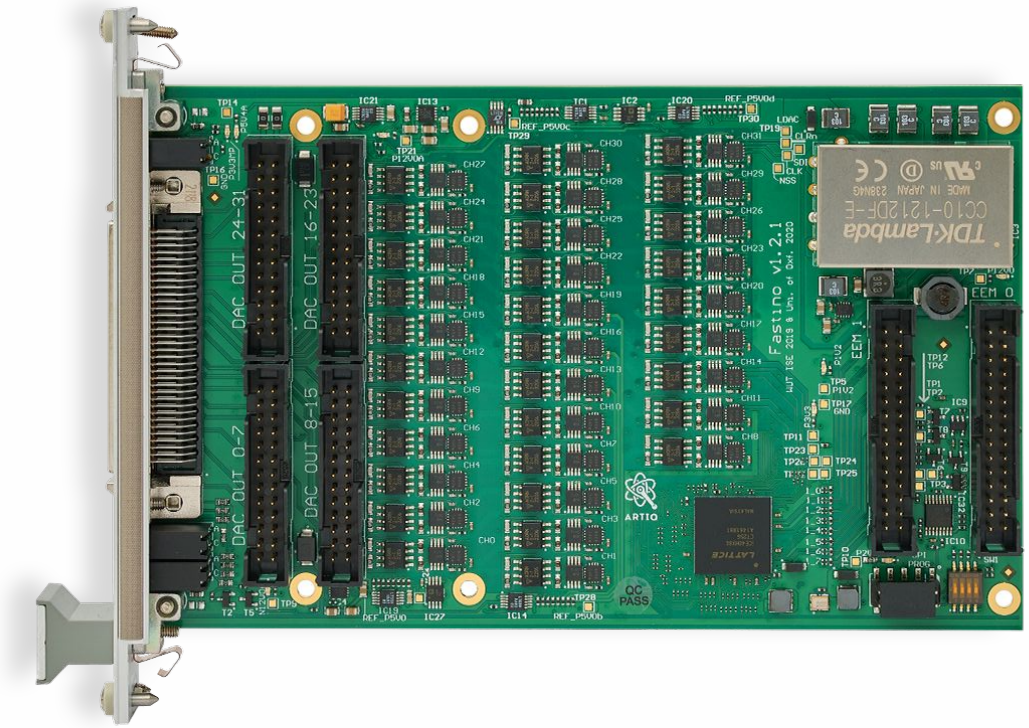
Warsaw University of Technology

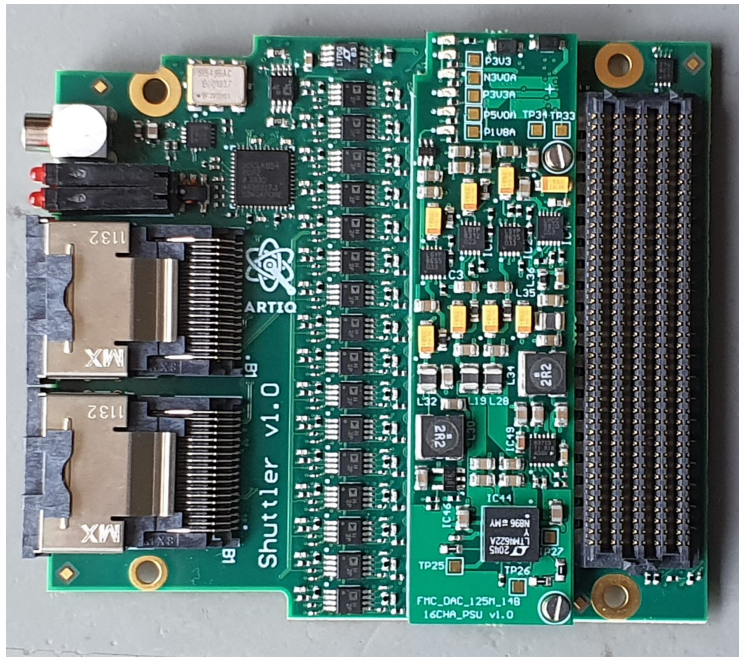
Use cases

Frame Grabber + Camera

Use cases

Fastino or Shuttler

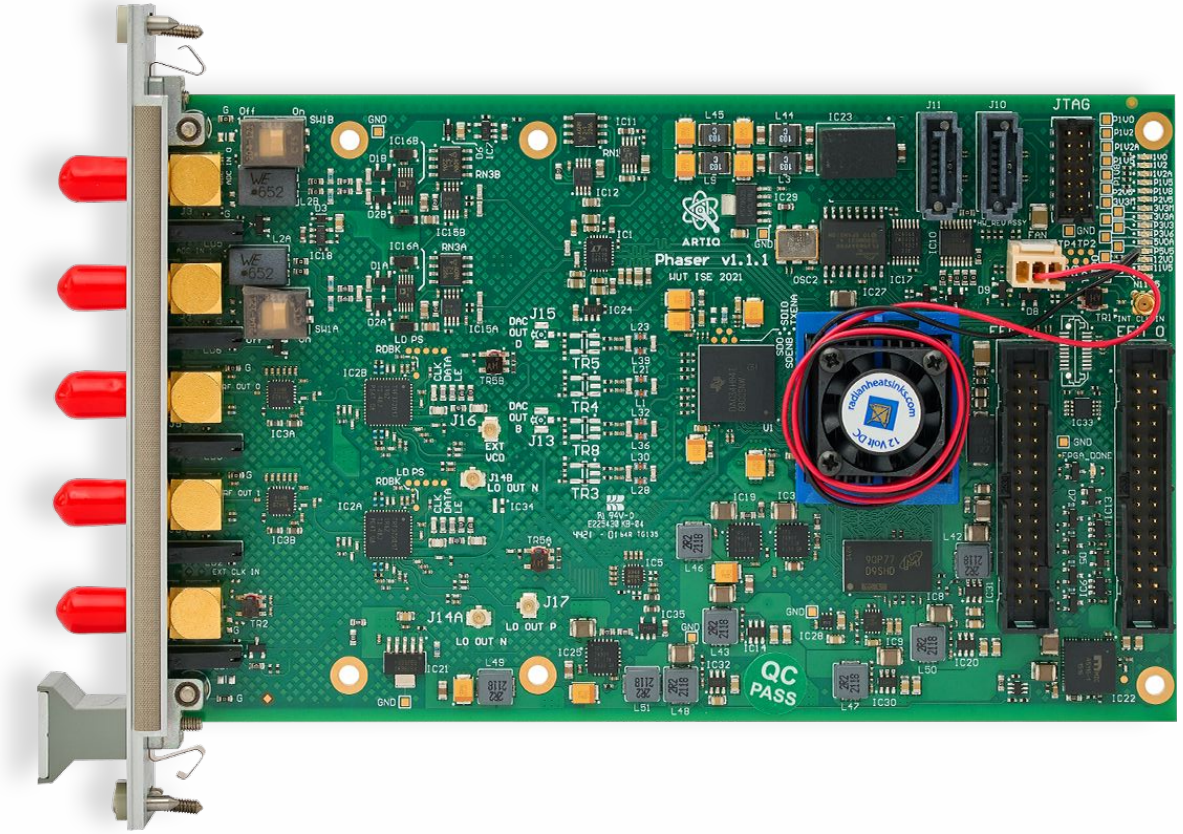




Warsaw University of Technology

Use cases

Phaser



Warsaw University
of Technology

Warsaw University of Technology

https://github.com/elhep/artiq_dax_tutorial_materials

- Paweł Kulik
- Mikołaj Sowiński
- Jakub Matyas

If you have any questions, feel free to contact us:

- pawel.kulik@pw.edu.pl
- mikolaj.sowinski@pw.edu.pl