

**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](https://github.com/elhep/artiq_dax_tutorial_materials)

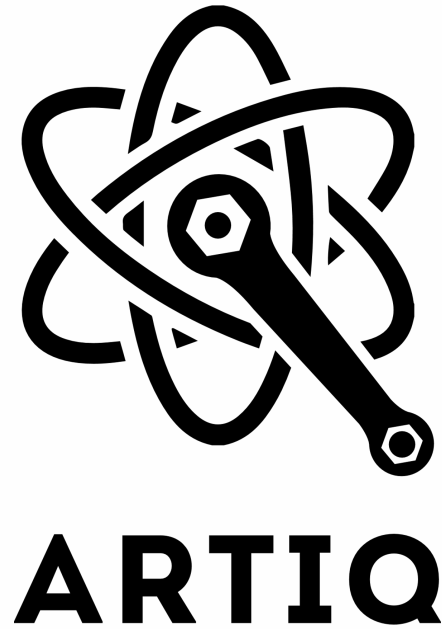
ARTIQ 8 manual (used for this tutorial):

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



# Glossary

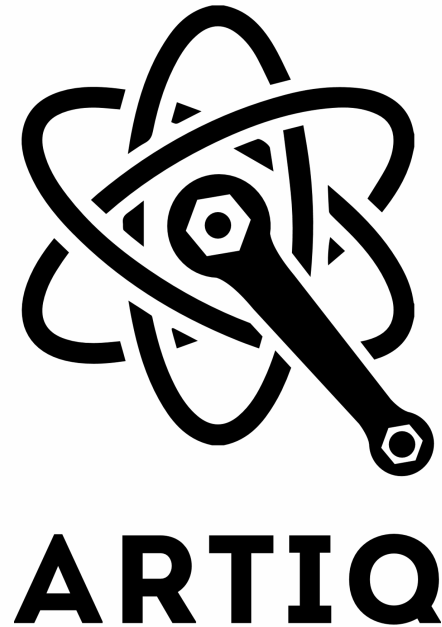
- ARTIQ - Advanced Real-Time Infrastructure for Quantum physics



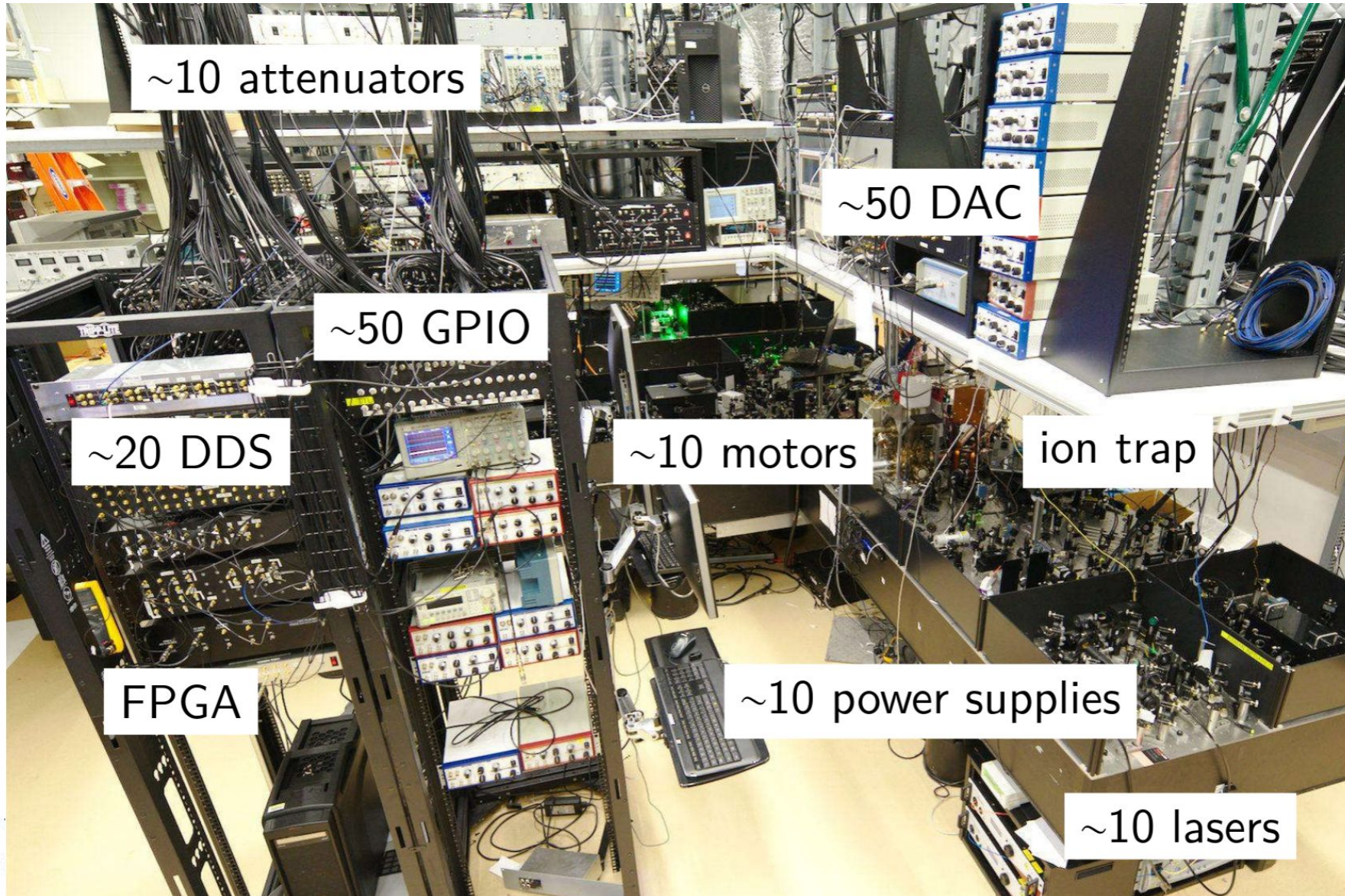


# Motivation for ARTIQ

- 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



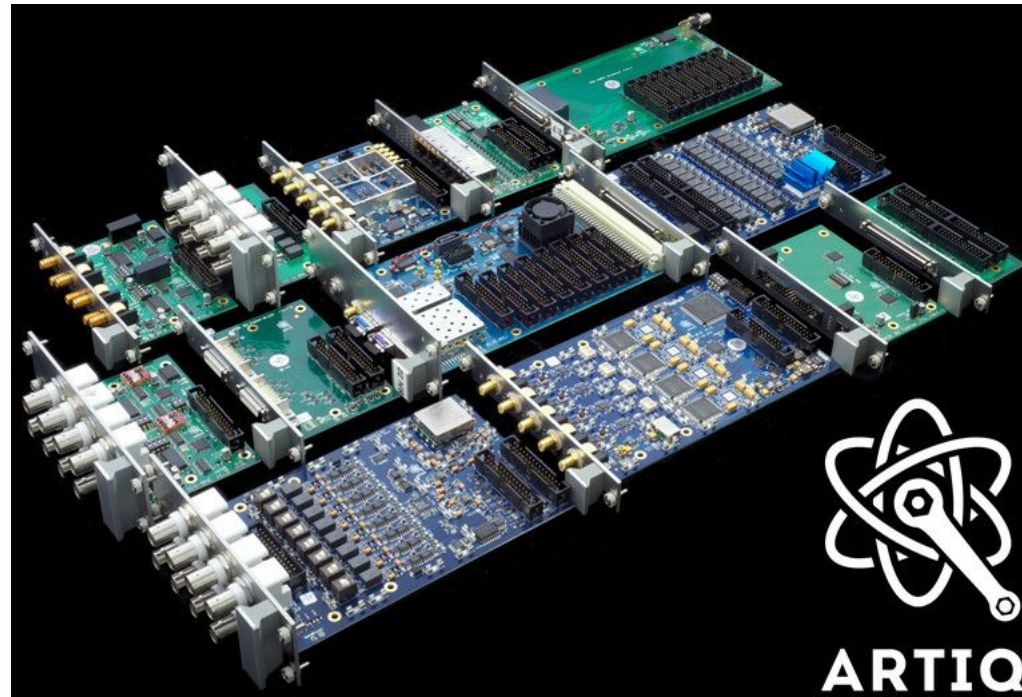
# Motivation for ARTIQ





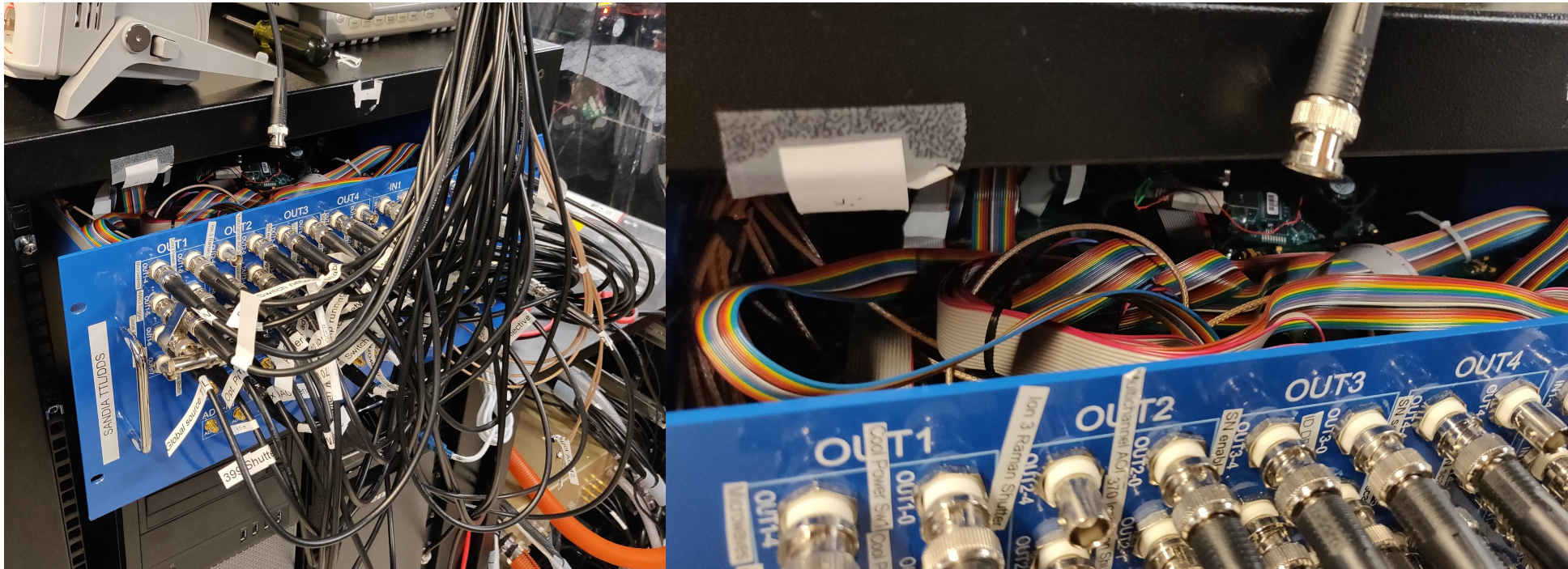
# Glossary

- ARTIQ
- Sinara - Hardware designed for ARTIQ



# Motivation for Sinara

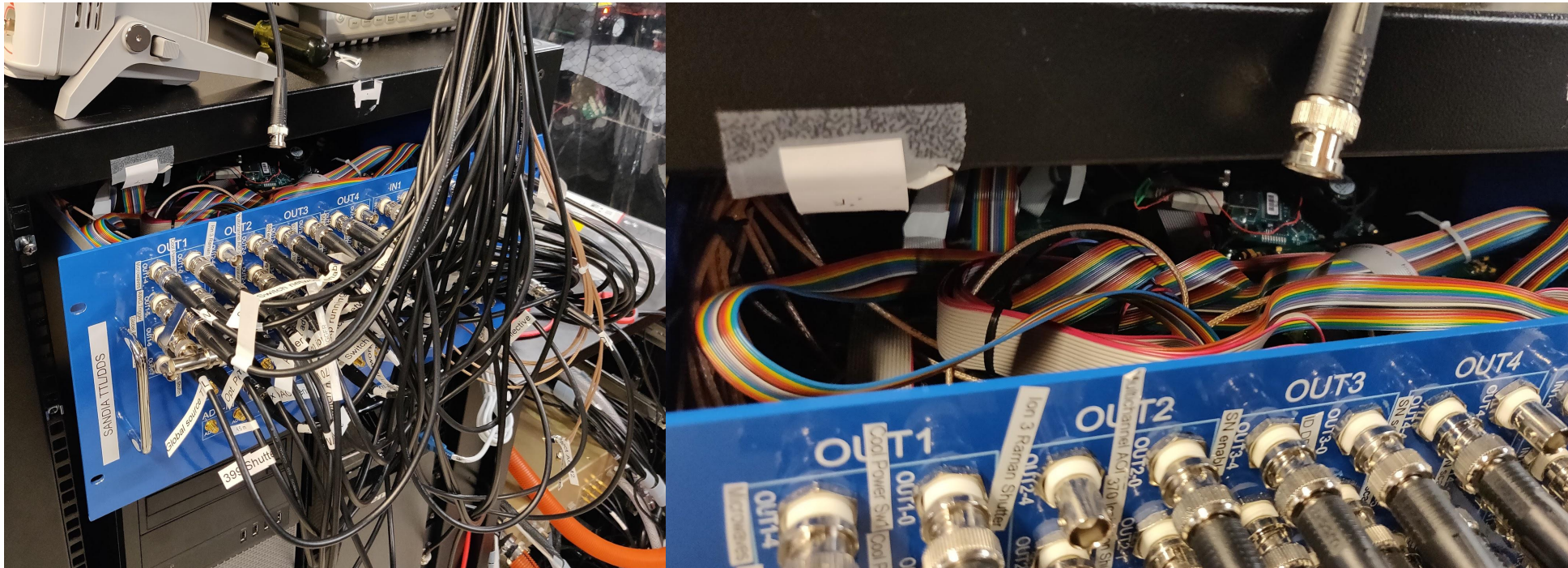
- Physicists are not engineers - and that's OK!
- If you're not careful, you'll end up with this:





# 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



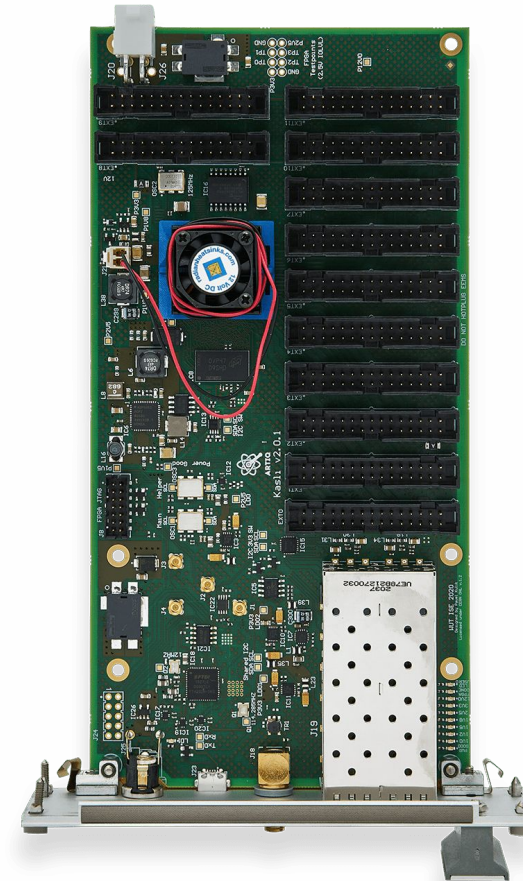
# Motivation for Sinara

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



# Glossary

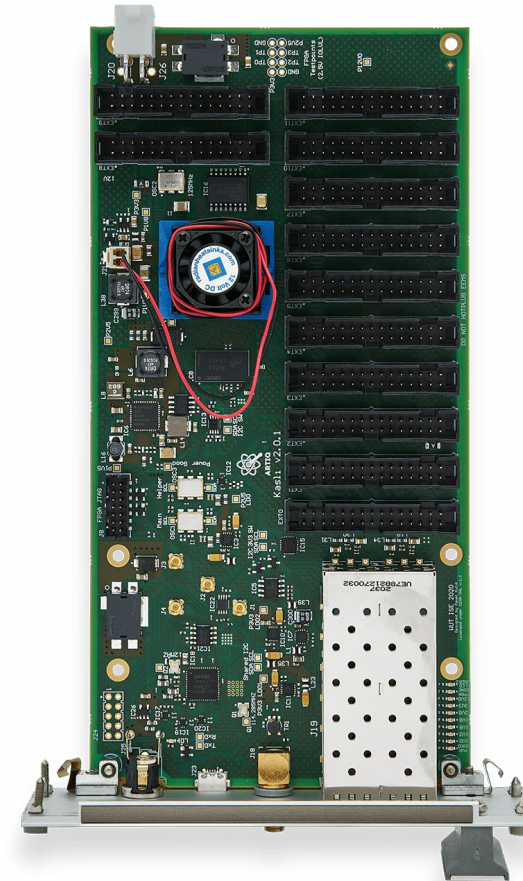
- ARTIQ
- Sinara
- Controller





# Glossary

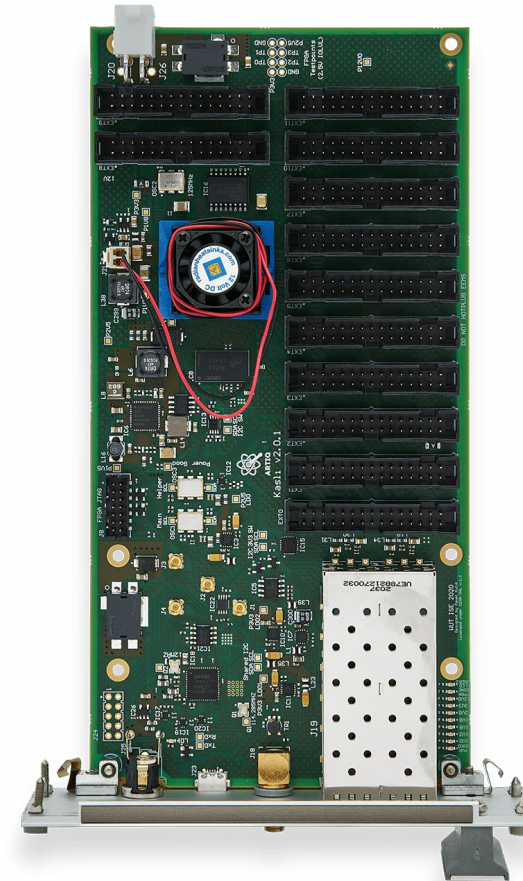
- ARTIQ
- Sinara
- Controller
- Satellite - secondary controller





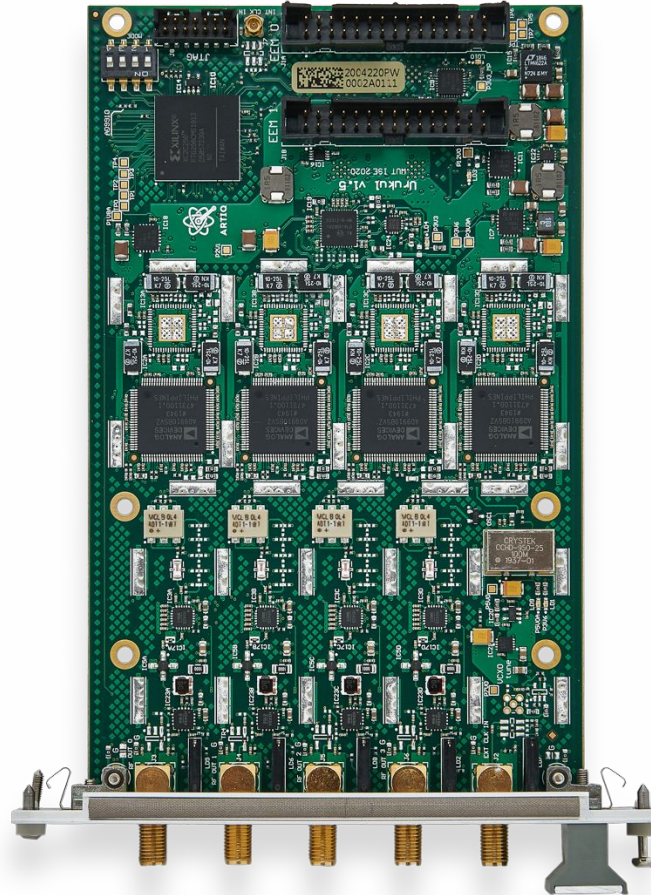
# Glossary

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



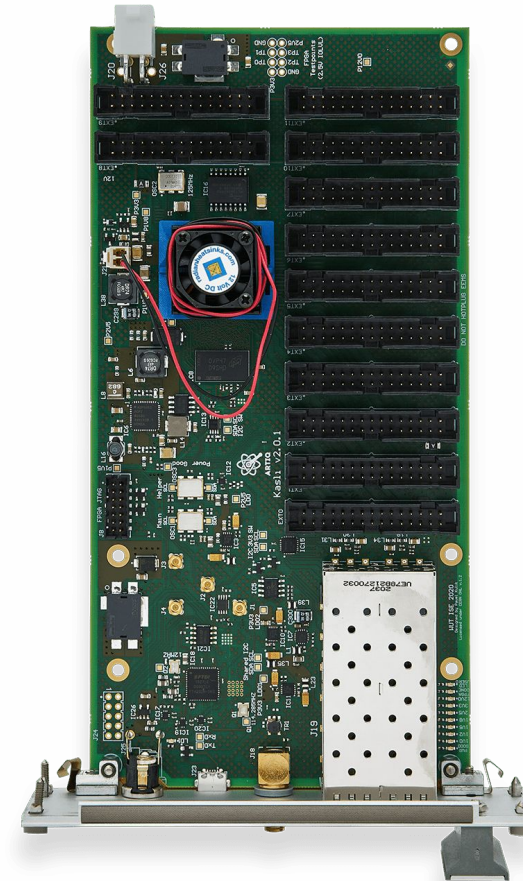
# Glossary

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



# Glossary

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



# 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):

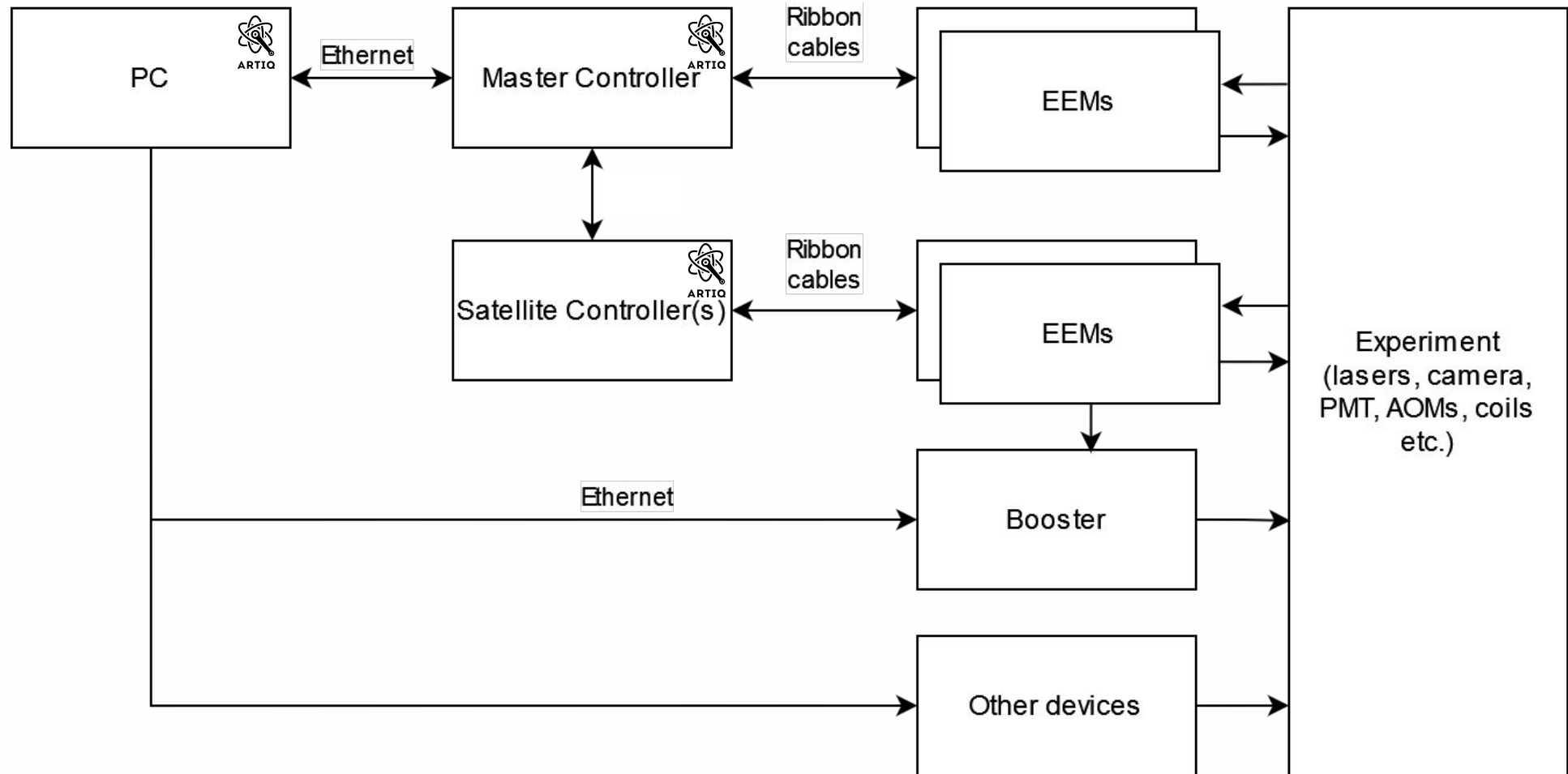
    def build(self):
        self.setattr_device("core")
        self.setattr_device("ttl1")

    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)
```

# ARTIQ system architecture

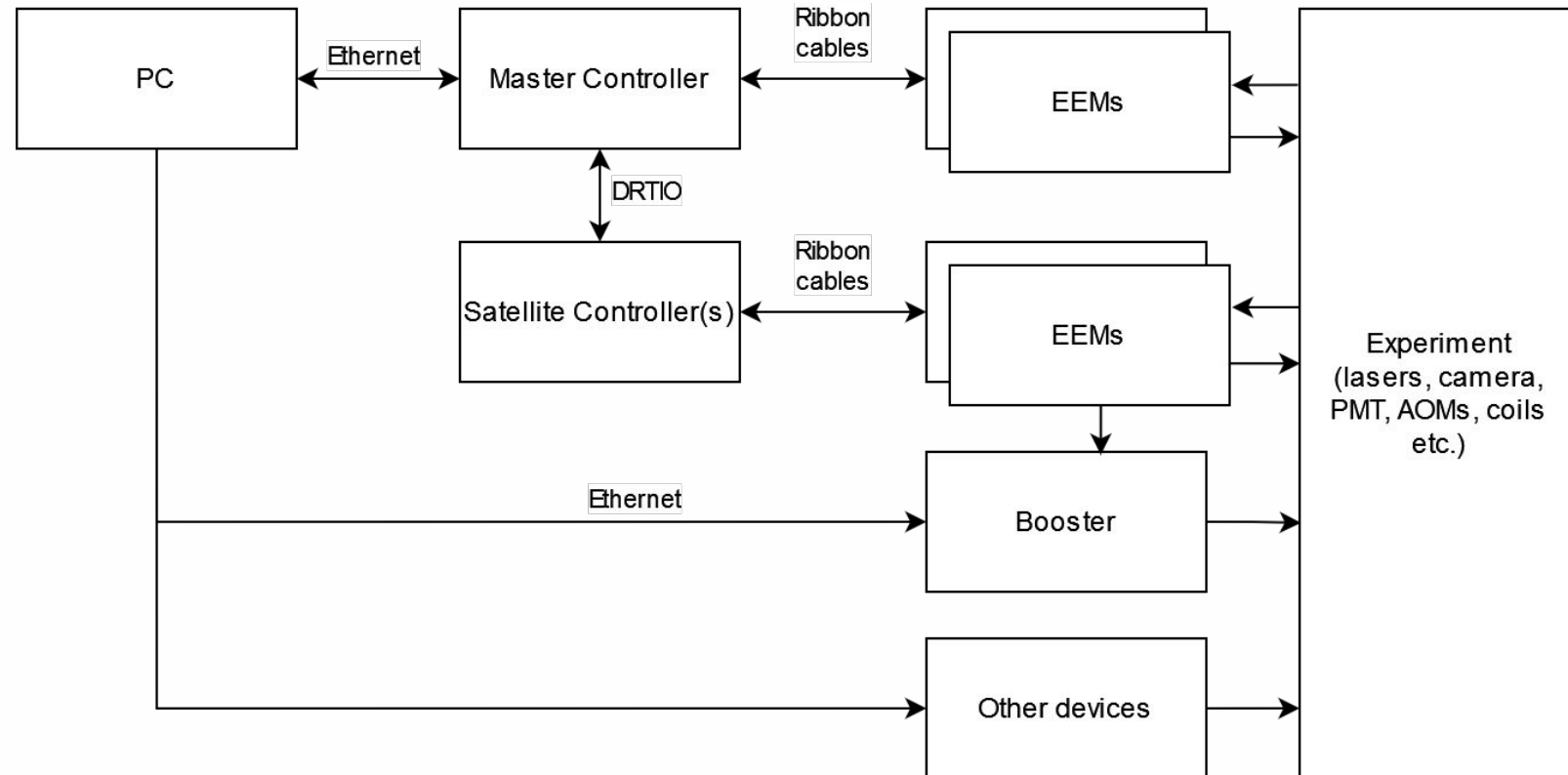


# What happens with my code?

@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

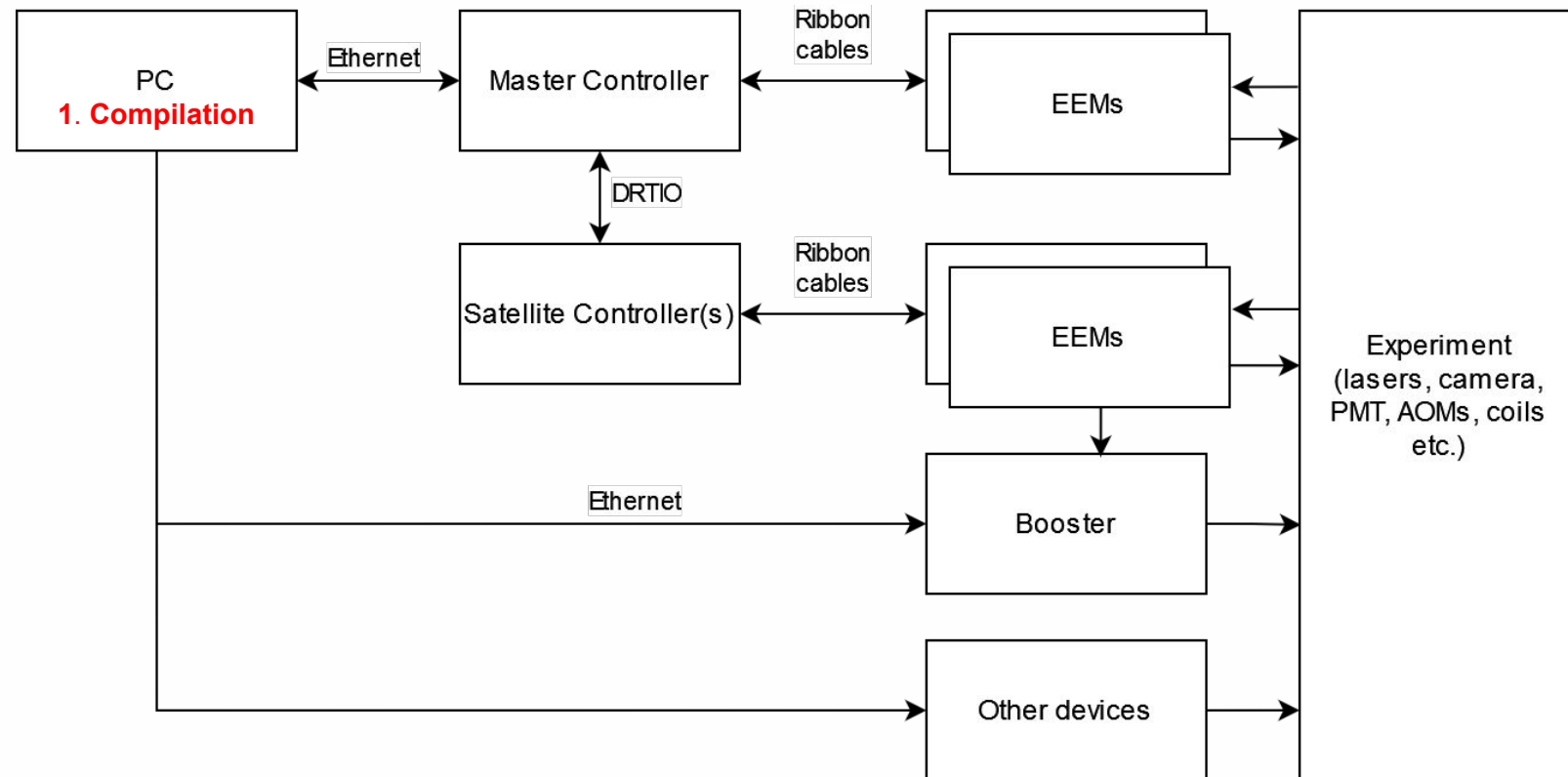


# What happens with my code?

@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



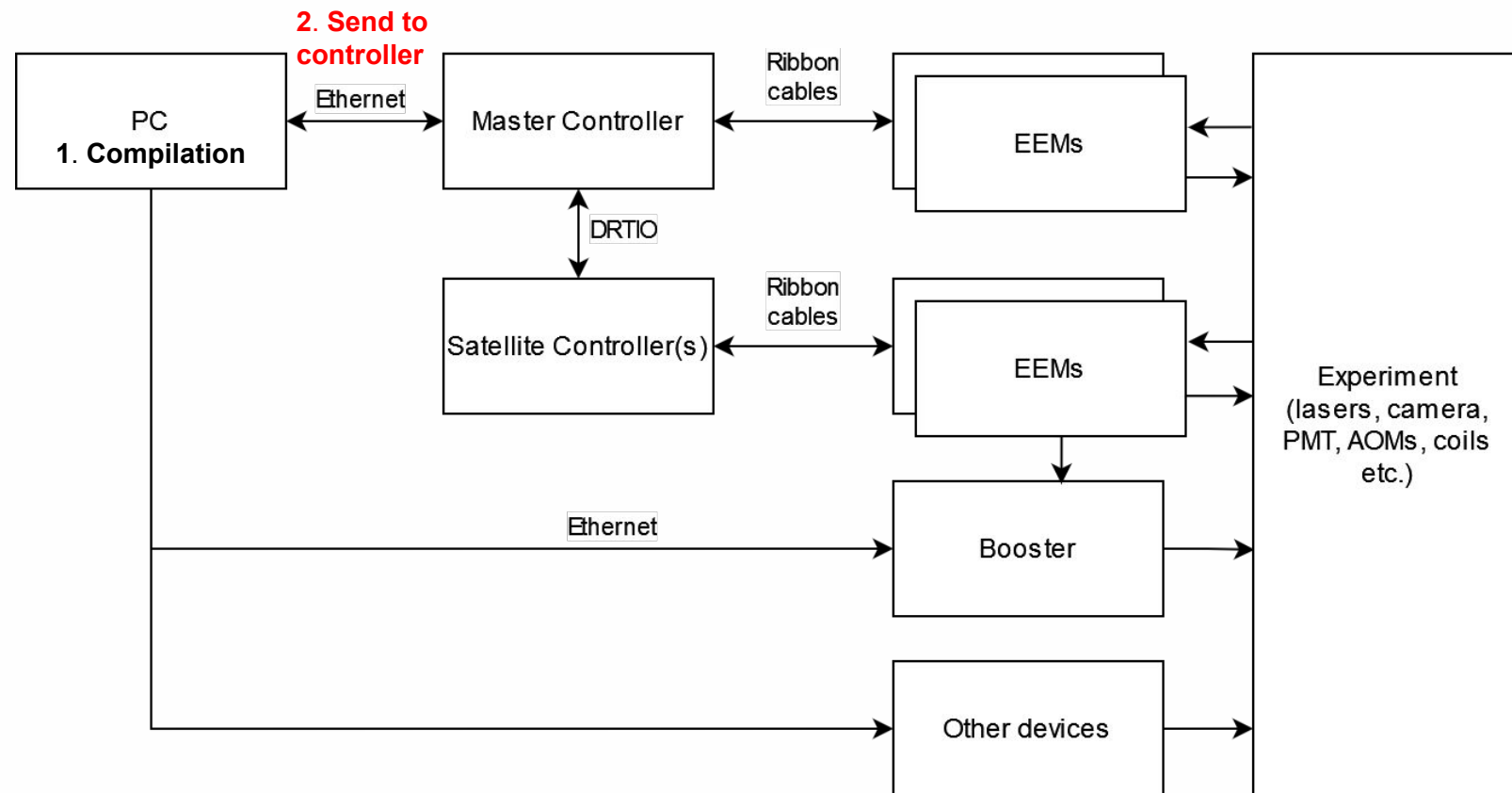


# What happens with my code?

@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



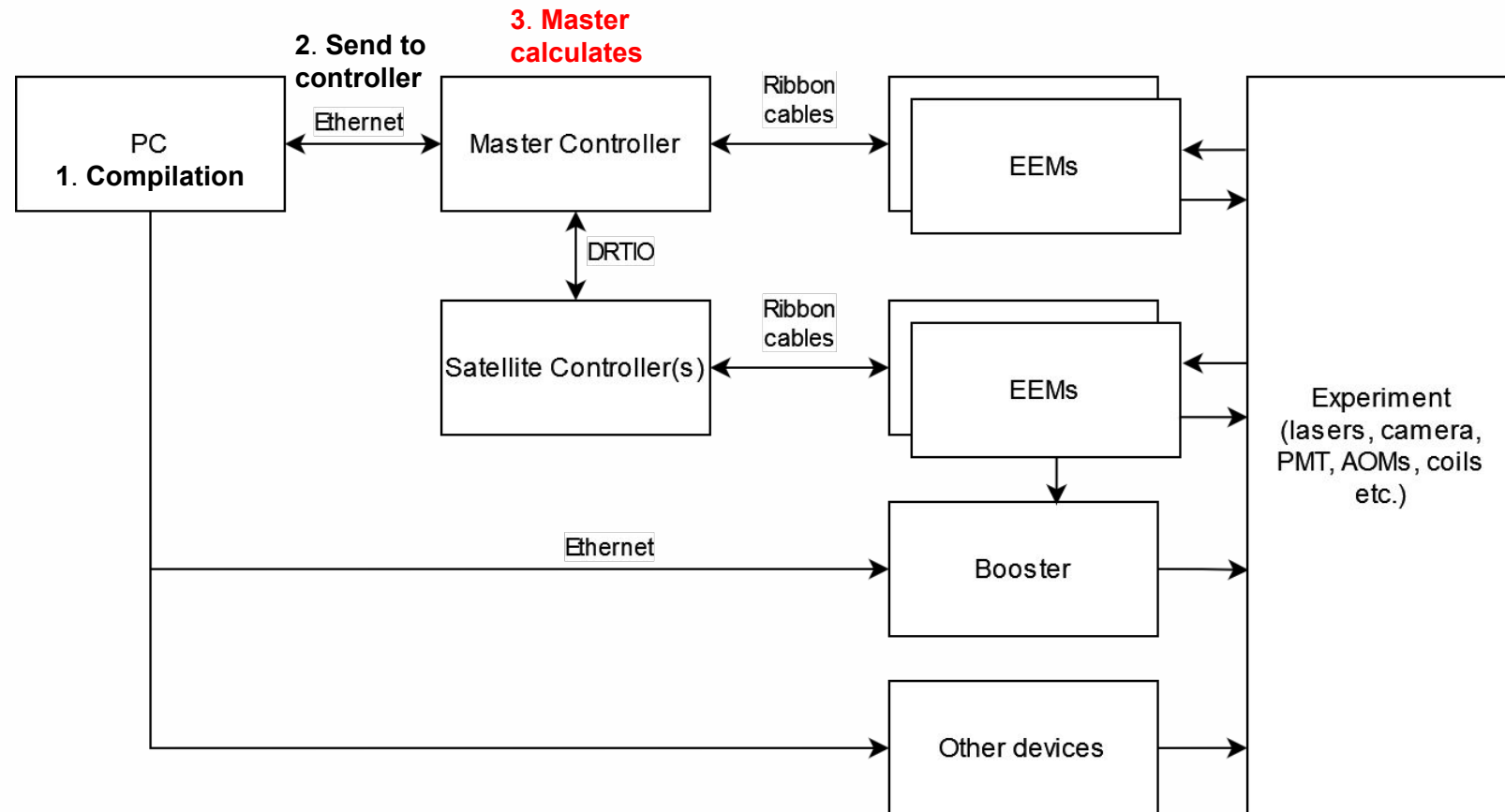


# What happens with my code?

@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

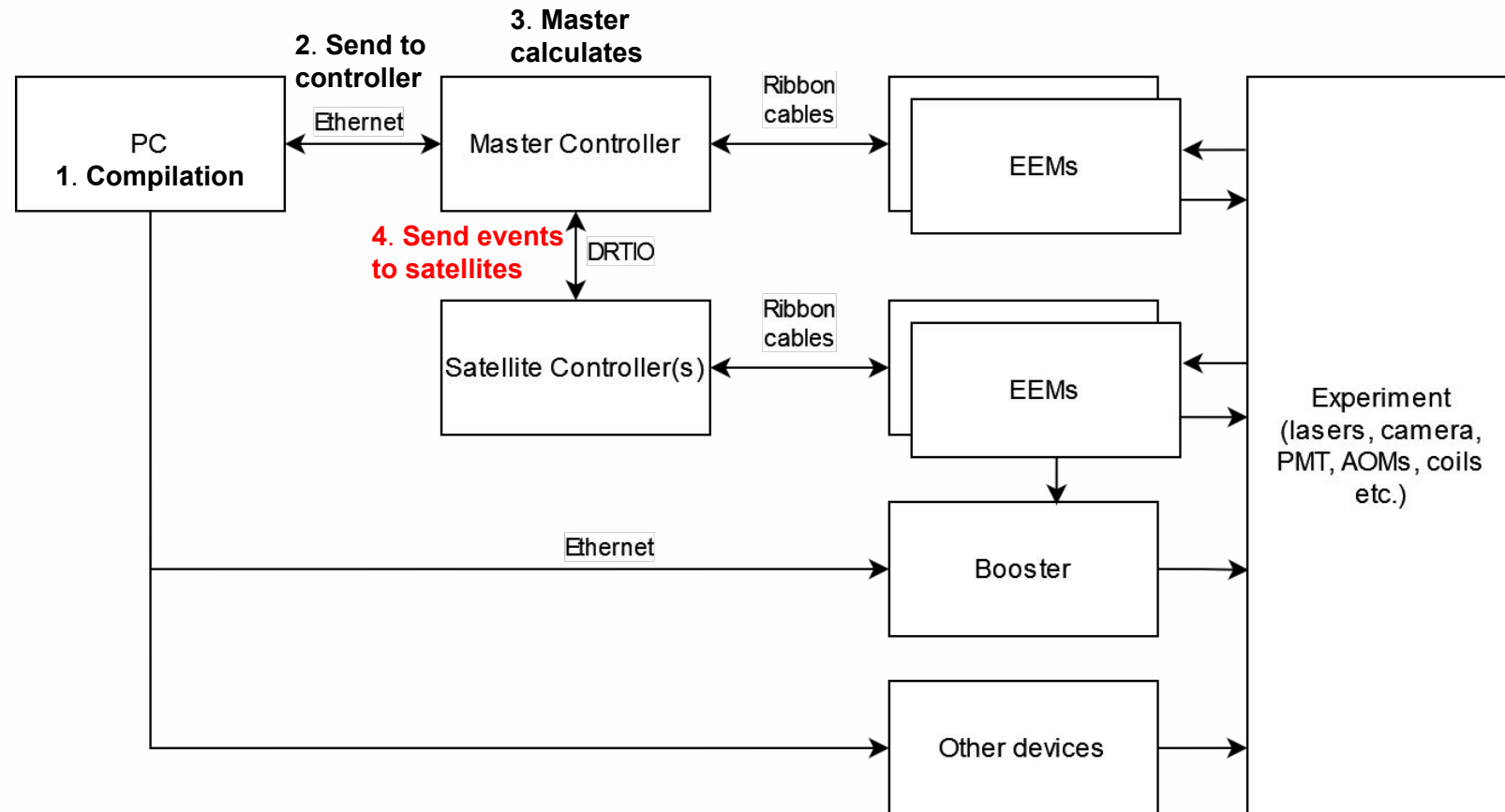


# What happens with my code?

@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

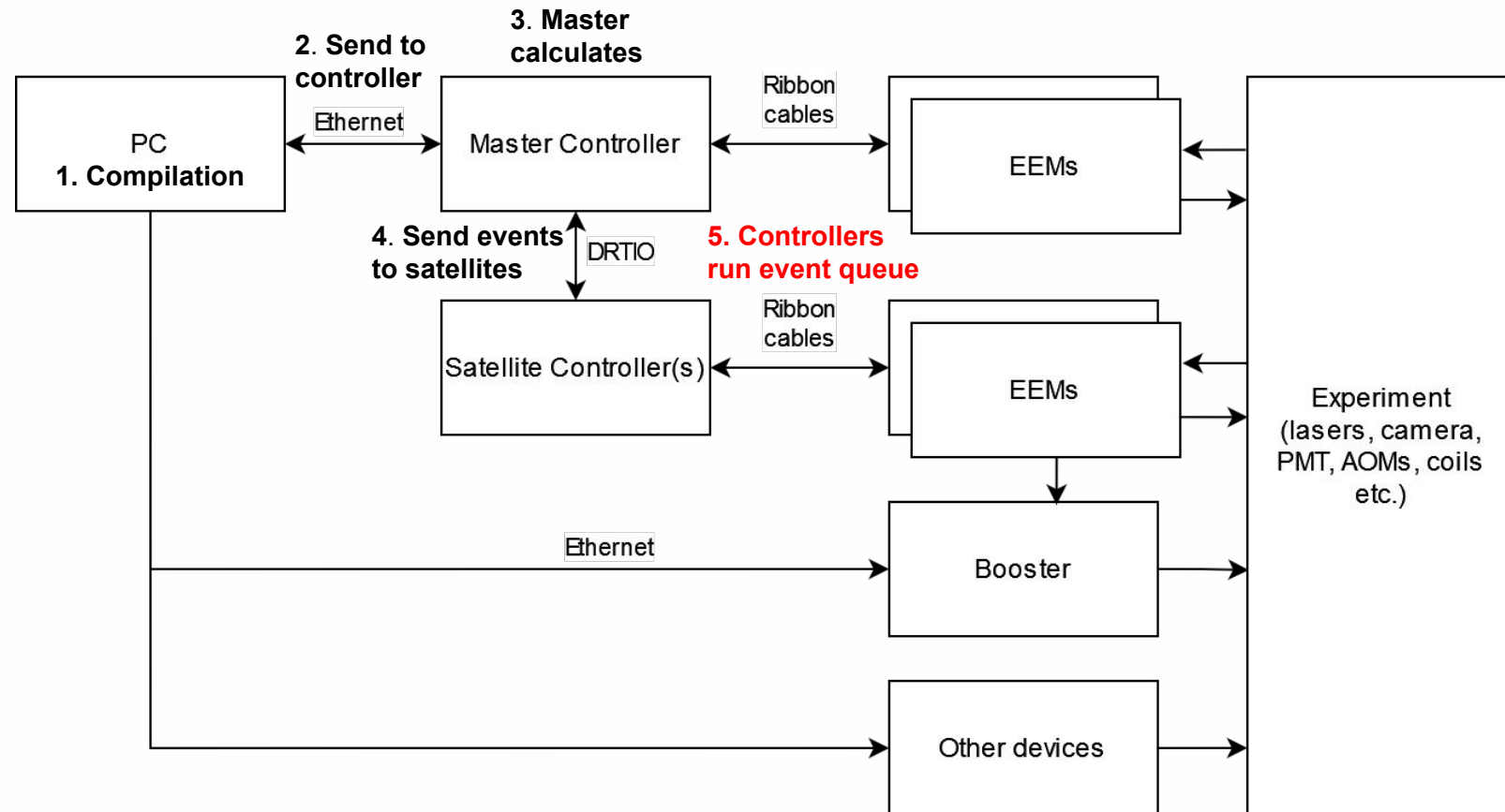


# What happens with my code?

@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

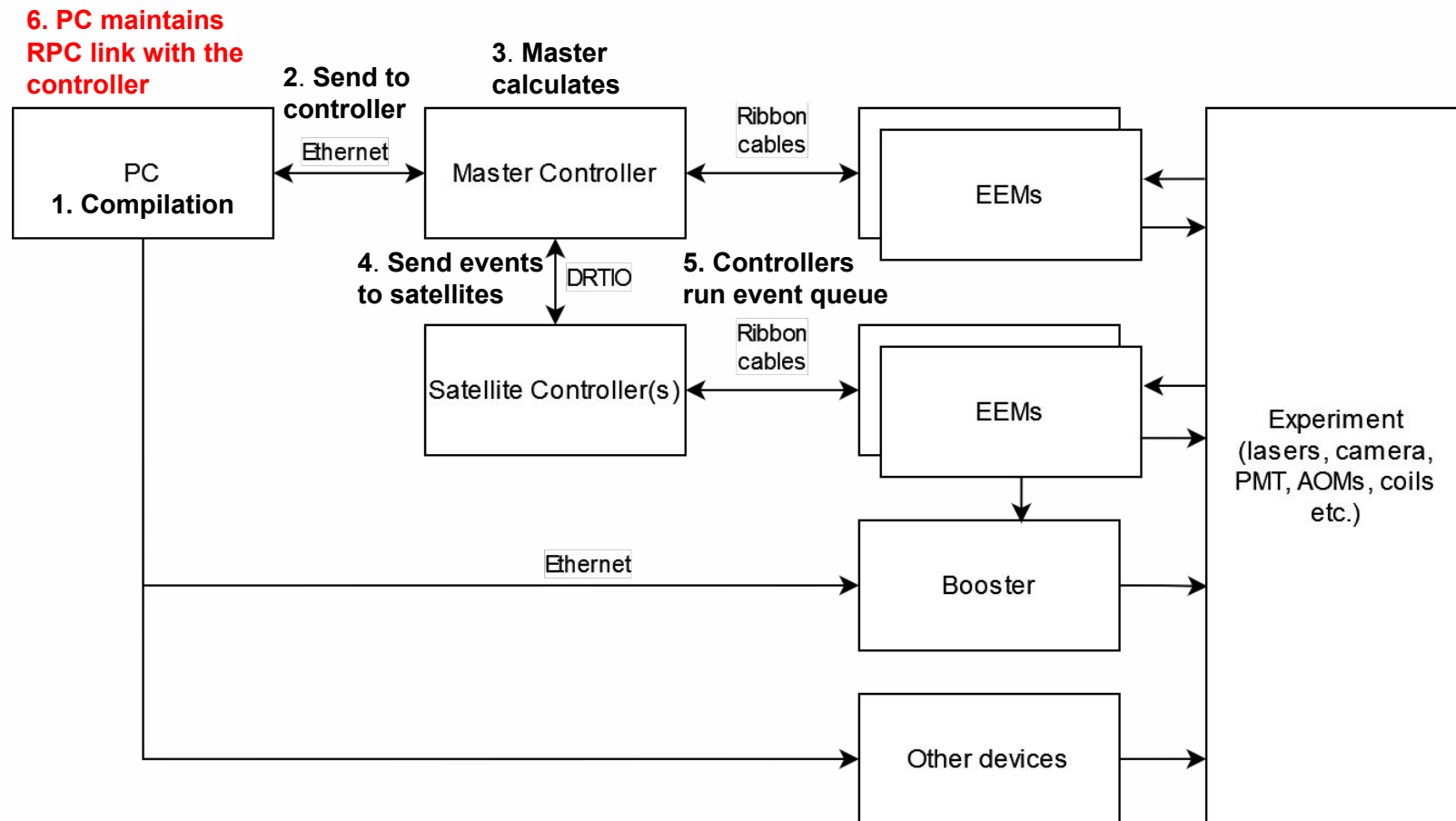


# What happens with my code?

@kernel

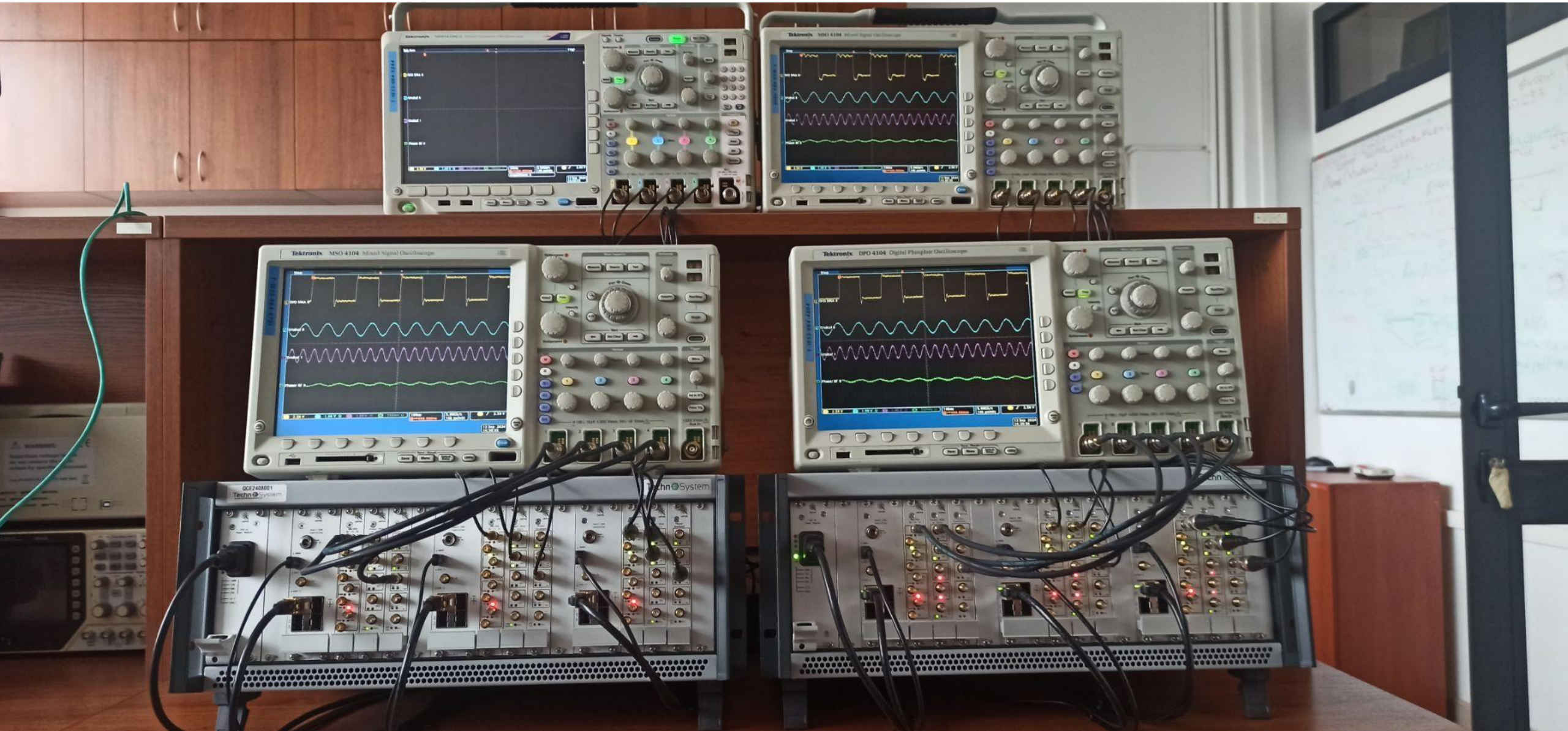
```
def run(self):  
    self.core.reset()  
    self.ttl1.pulse(100*ns)
```

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

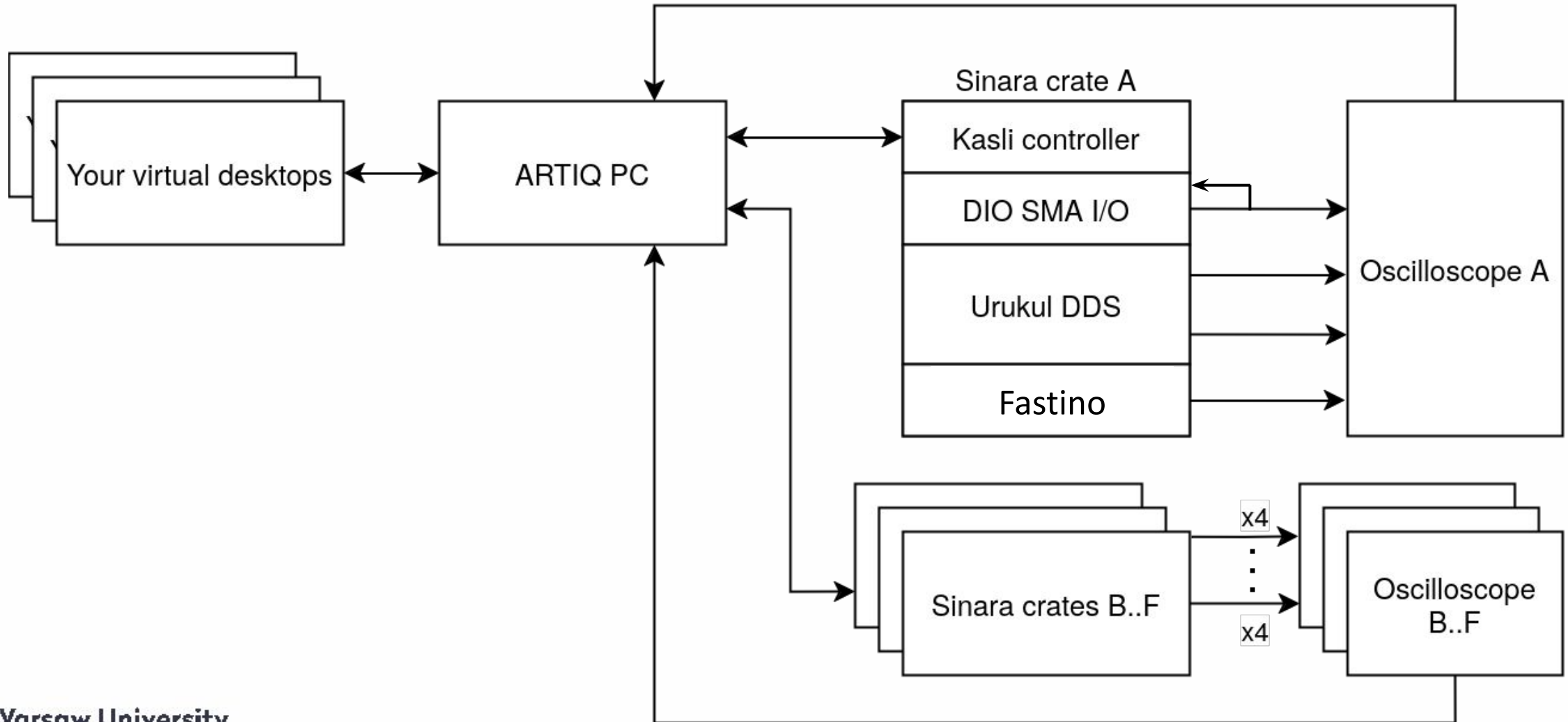




# Devices that will be used

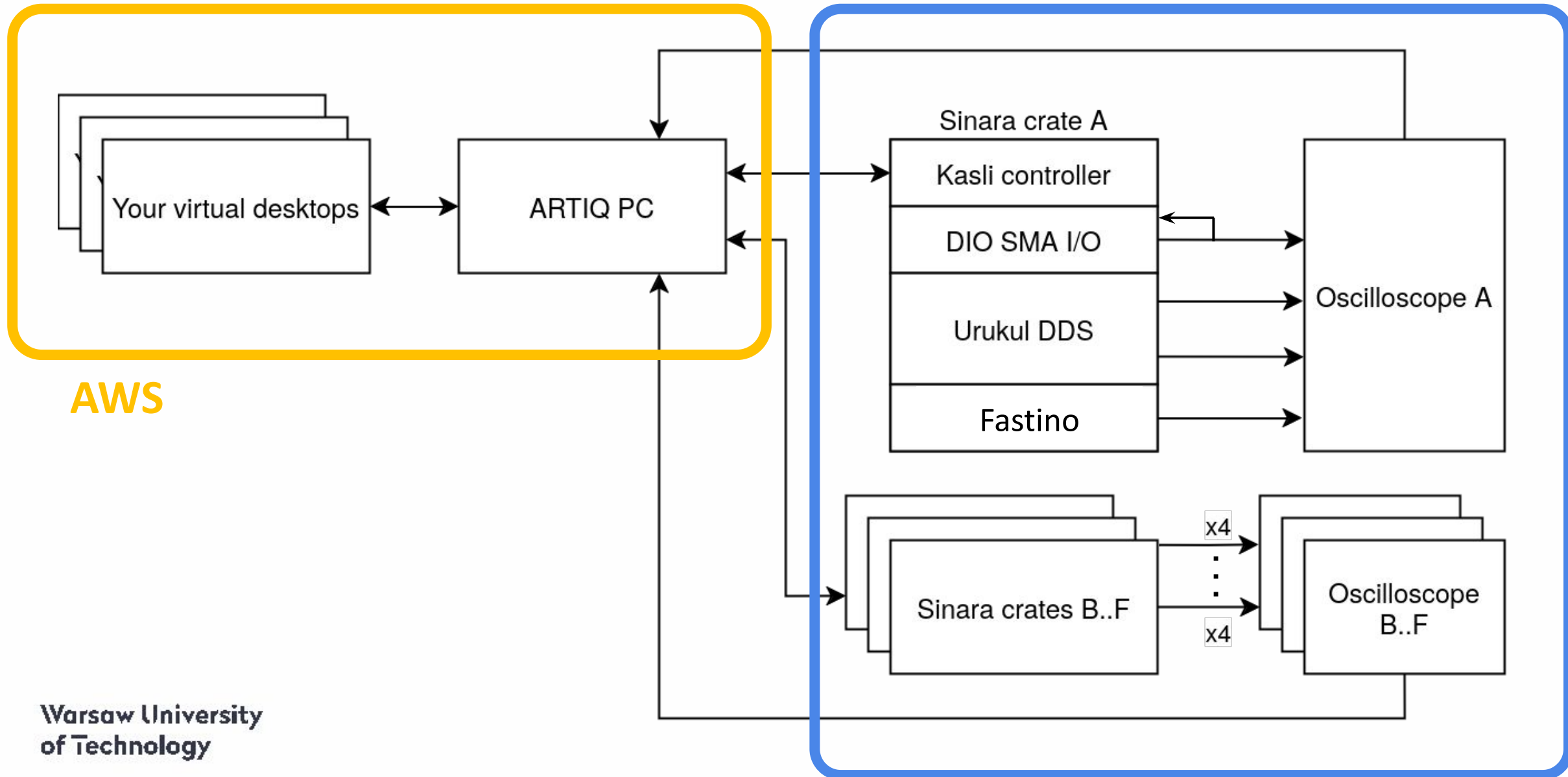


# Devices that will be used



# Devices that will be used

Warsaw Lab

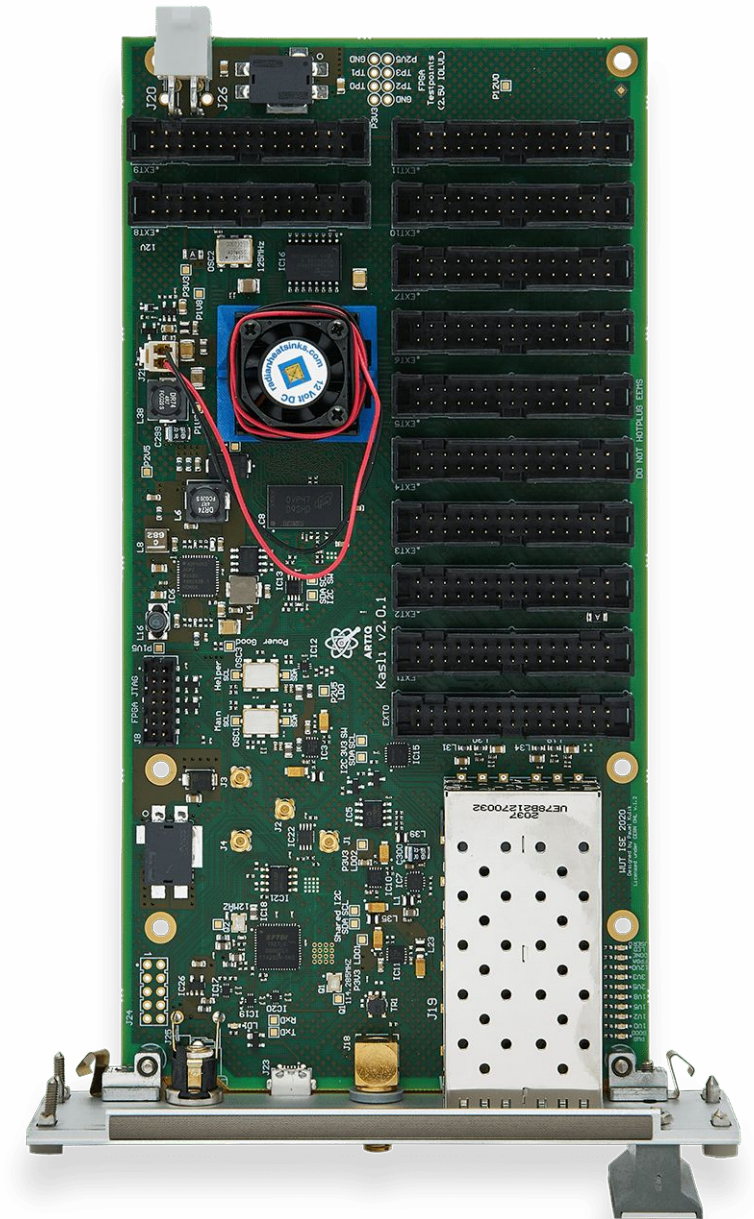


AWS



# Devices that will be used

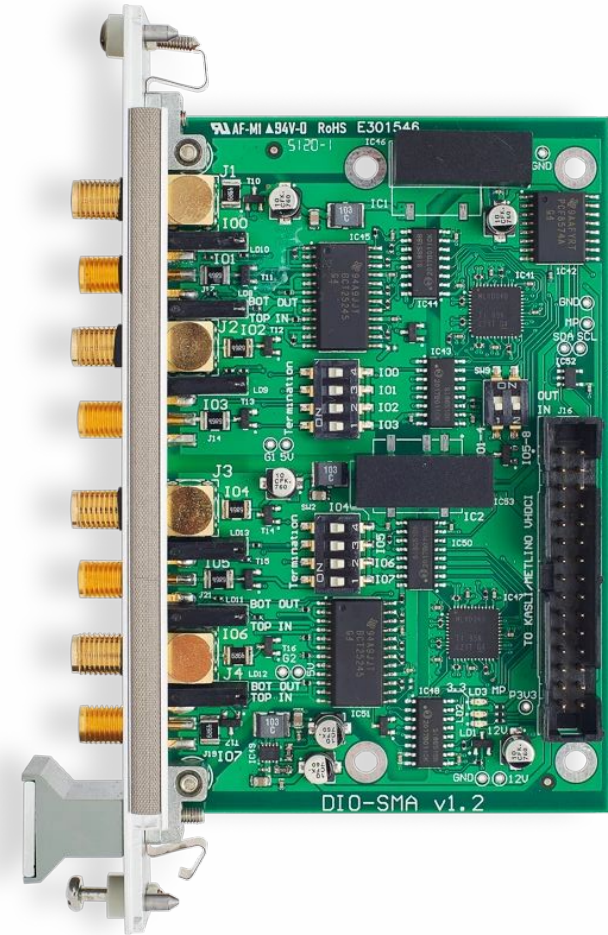
- Kasli
  - AMD Artix-7 based controller
  - up to 12 peripherals
  - up to 3 downstream satellites
  - can be both master and satellite
  - <https://github.com/sinara-hw/Kasli>





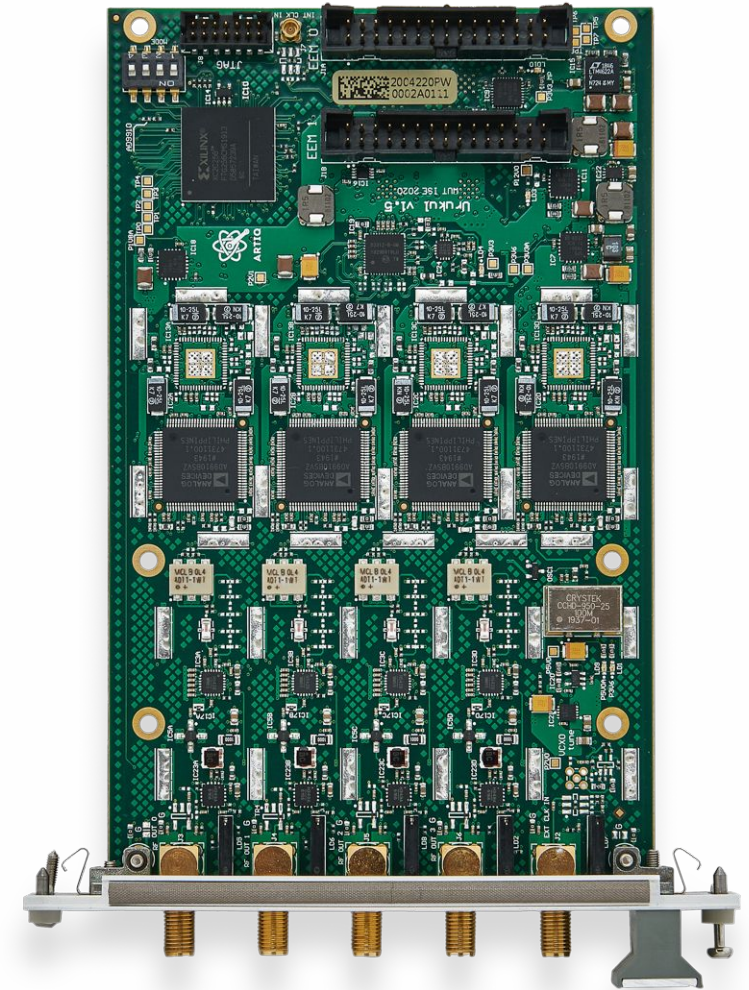
# Devices that will be used

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



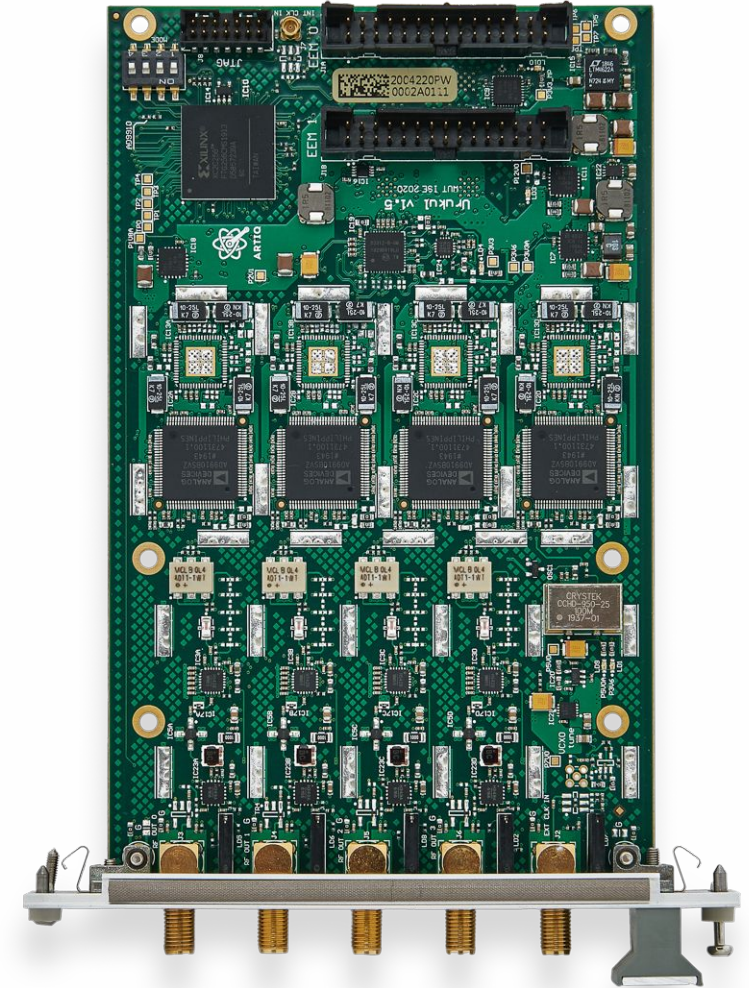
# Devices that will be used

- 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>



# Devices that will be used

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



# Organization

Connect to virtual desktop using your personal link



# Organization

1. Connect to virtual desktop using your personal link



## Choose your application to get started



Desktop



ARTIQ

[Sinara Home Page](#)



# Organization

The image shows a Linux desktop environment with a dark theme. At the top left, the 'Applications' menu is highlighted with a red arrow labeled '1'. In the top center, the system clock shows 'Sun 21:46'. At the top right, there are icons for network, volume, and power. A search bar with the text 'Type to search...' is located below the clock. The 'creotech' logo is visible in the top right corner. The main area displays a grid of application icons. A red box highlights a row of six 'Dashboard' icons (A through F), with a red arrow labeled '4' pointing to the right side of this row. Below this row are icons for 'Files', 'GTKWave', 'Settings', 'Terminal', 'Text Editor', and 'VSCodium'. A red arrow labeled '3' points to the 'VSCodium' icon. On the left side, a vertical dock contains icons for 'Files', 'Terminal', and a grid icon. A red arrow labeled '2' points to the grid icon in the dock. At the bottom center, there are two buttons labeled 'Frequent' and 'All'.

Applications

Sun 21:46

Type to search...

creotech

Dashboard A Dashboard B Dashboard C Dashboard D Dashboard E Dashboard F

Files GTKWave Settings Terminal Text Editor VSCodium

Frequent All

1

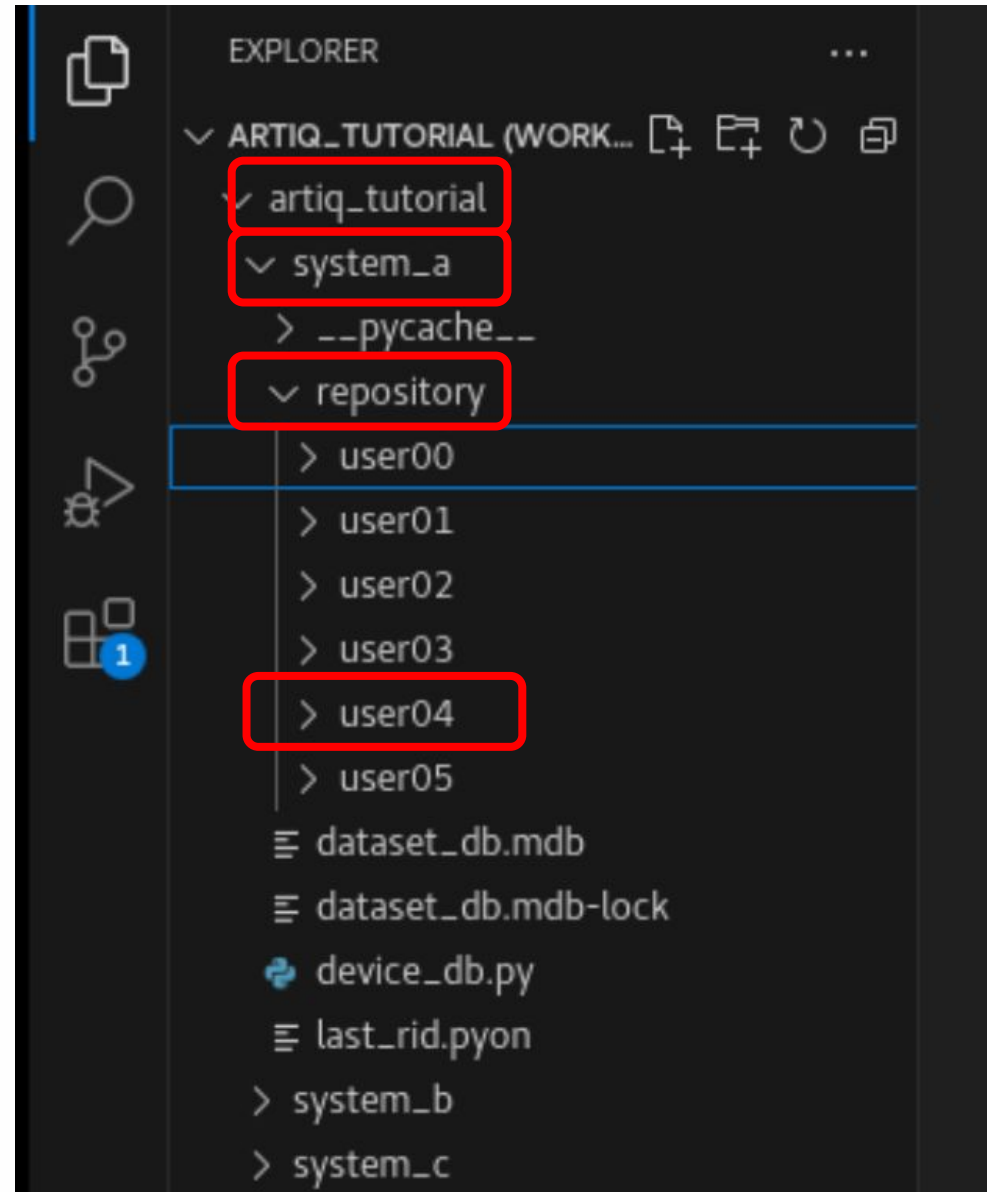
2

3

4

# Organization


**artiq\_tutorial**  
**system\_<a..f>**  
**repository**  
**user<00..05>**



# Organization

Applications ARTIQ Dashboard - System A Wed 14:56

ARTIQ Dashboard - System A



Applet: User 5 Scope View

Name	Command
<input type="checkbox"/> User 0 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 1 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 2 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 3 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 4 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input checked="" type="checkbox"/> User 5 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 6 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 7 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 8 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 9 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 10 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 11 Scope View	python /nix/artiq_dax_tutorial/applets/sho

Explorer Shortcuts TTL DDS DAC Datasets Applets

RID	Pipeline	Status	Prio	Due date	Revision	File	Class name
-----	----------	--------	------	----------	----------	------	------------


Schedule Log



# Organization

Applications ARTIQ Dashboard - System A Wed 14:56

ARTIQ Dashboard - System A



Applet: User 5 Scope View

Name	Command
<input type="checkbox"/> User 0 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 1 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 2 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 3 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 4 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input checked="" type="checkbox"/> User 5 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 6 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 7 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 8 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 9 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 10 Scope View	python /nix/artiq_dax_tutorial/applets/sho
<input type="checkbox"/> User 11 Scope View	python /nix/artiq_dax_tutorial/applets/sho

Explorer Shortcuts TTL DDS DAC Datasets Applets

RID	Pipeline	Status	Prio	Due date	Revision	File	Class name
-----	----------	--------	------	----------	----------	------	------------

Schedule Log

# Organization

ARTIQ Dashboard - System A

Applet: User 0 Scope View

repo:user03/Initialize

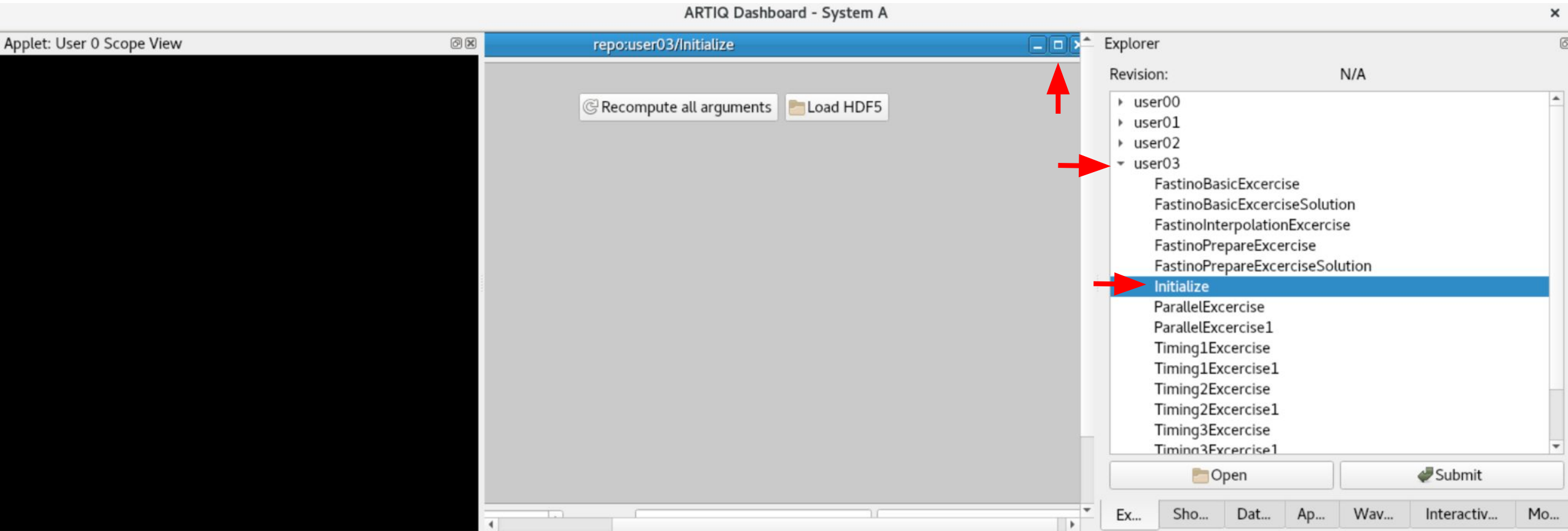
Recompute all arguments Load HDF5

Revision: N/A

- user00
- user01
- user02
- user03
  - FastinoBasicExcercise
  - FastinoBasicExcerciseSolution
  - FastinoInterpolationExcercise
  - FastinoPrepareExcercise
  - FastinoPrepareExcerciseSolution
  - Initialize**
  - ParallelExcercise
  - ParallelExcercise1
  - Timing1Excercise
  - Timing1Excercise1
  - Timing2Excercise
  - Timing2Excercise1
  - Timing3Excercise
  - Timing3Excercise1

Open Submit

Ex... Sho... Dat... Ap... Wav... Interactiv... Mo...



Schedule

RID	Pipeline	Status	Prio	Due date	Revision	File	Class name
-----	----------	--------	------	----------	----------	------	------------

# Organization

The screenshot displays the ARTIQ Dashboard interface for 'System A'. The main workspace is titled 'repo:user03/artiq/Initialize' and contains a 'Recompute all arguments' button and a 'Load HDF5' button. Below the workspace are controls for 'Due date' (Sep 19 2024 00:00:00), 'Pipeline' (ain), 'Priority' (0), 'Logging level' (WARNING), and 'Revision' (c...). A 'Submit' button is also present. On the right, the 'Explorer' panel shows a tree view with 'Initialize' selected under 'artiq'. The 'Log' panel at the bottom has a 'Minimum level' dropdown set to 'ERROR' and a search field containing 'user03'. A red arrow points from the 'user03' text in the workspace to the search field, and another red arrow points from the search field to the 'Log' panel. The 'Log' panel is currently empty.

Applications ARTIQ Dashboard - System A Thu 13:26

Applet: User 3 Scope View

repo:user03/artiq/Initialize

No arguments

Recompute all arguments Load HDF5

Due date: Sep 19 2024 00:00:00 Pipeline: ain Submit

Priority: 0 Flush

Logging level: WARNING Revision: c... Terminate instances

Revision: N/A

- user00
- user01
- user02
- user03
  - artiq
    - DMAExercise
    - Initialize
    - ParallelExercise
    - Timing1Exercise
    - Timing2Exercise
  - artiq\_solutions
  - dax
- user04
- user05
- user06
- user07
- user08
- user09
- user10

Open Submit

Explorer Shortcuts TTL DDS DAC Datasets Applets

Log

Minimum level: ERROR user03

Source Message

Schedule Log

# Organization

Applications ARTIQ Dashboard - System A Thu 13:33

ARTIQ Dashboard - System A

Applet: User 3 Scope View

repo:user03/artiq/Initialize

No arguments

Recompute all arguments Load HDF5

Submit

Due date: Sep 19 2024 00:00:00 Pipeline: ain Priority: 0 Logging level: WARNING Revision: c... Terminate instances

Revision: N/A

- user01
- user02
- user03
  - artiq
    - DMAExercise
    - Initialize
    - ParallelExercise
    - Timing1Exercise
    - Timing2Exercise
  - artiq\_solutions
- dax
- user04
- user05
- user06
- user07
- user08
- user09
- user10
- user11

Open Submit

Explorer Shortcuts TTL DDS DAC Datasets Applets

Schedule

RID	Pipeline	Status	Prio	Due date	Revision	File	Class name
1	main	running	0		N/A	user03/artiq/user03_initialize.py	Initialize
2	main	prepare_done	0		N/A	user08/artiq/user08_initialize.py	Initialize
3	main	pending	0		N/A	user08/artiq/user08_initialize.py	Initialize

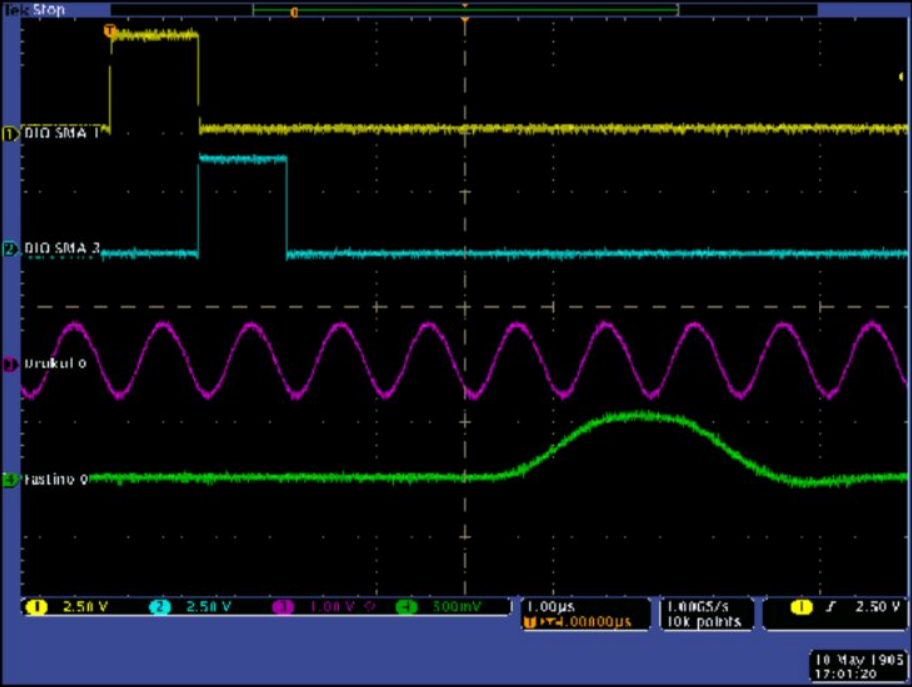
Schedule Log



# Organization

ARTIQ Dashboard - System A

Applet: User 0 Scope View



repo:user00/Initialize

No arguments

Recompute all arguments Load HDF5

Due date: Nov 3 2024 0 Pipeline: main Submit

Priority: 0 Flush Override d... Terminat

Logging level: WARNING Rev / ref: current

Explorer

Revision: N/A

- user00
  - FastinoBasicExercise
  - FastinoBasicExerciseSolution
  - FastinoInterpolationExercise
  - FastinoPrepareExercise
  - FastinoPrepareExerciseSolution
  - Initialize**
  - ParallelExercise
  - ParallelExercise1
  - Timing1Exercise
  - Timing1Exercise1
  - Timing2Exercise
  - Timing2Exercise1
  - Timing3Exercise
  - Timing3Exercise1
- user01
- user02
- user03

Open Submit

Ex... Sh... Da... Ap... Wa... In...

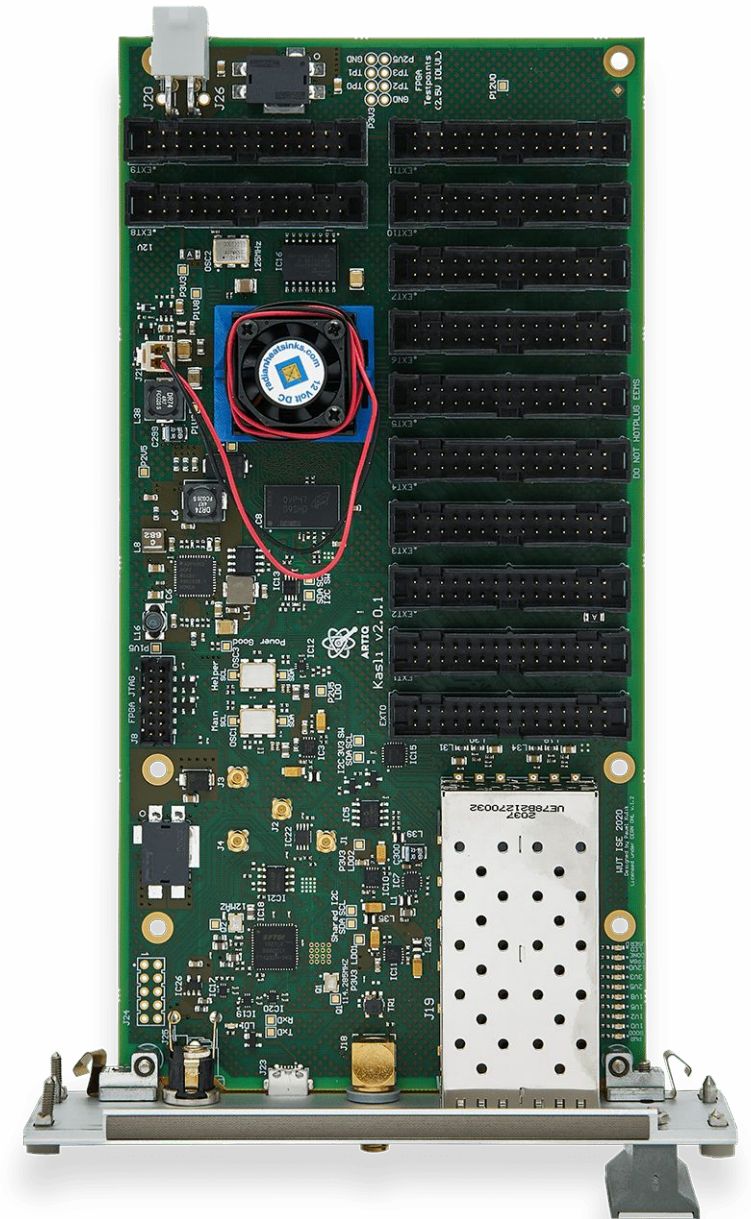
## Schedule

RID	Pipeline	Status	Prio	Due date	Revision	File	Class name
3	main	running	0		N/A	user03/user03_initialize.py	Initialize
4	main	prepare_done	0		N/A	user00/user00_initialize.py	Initialize

# ARTIQ DSL - what can be done in FPGA?

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

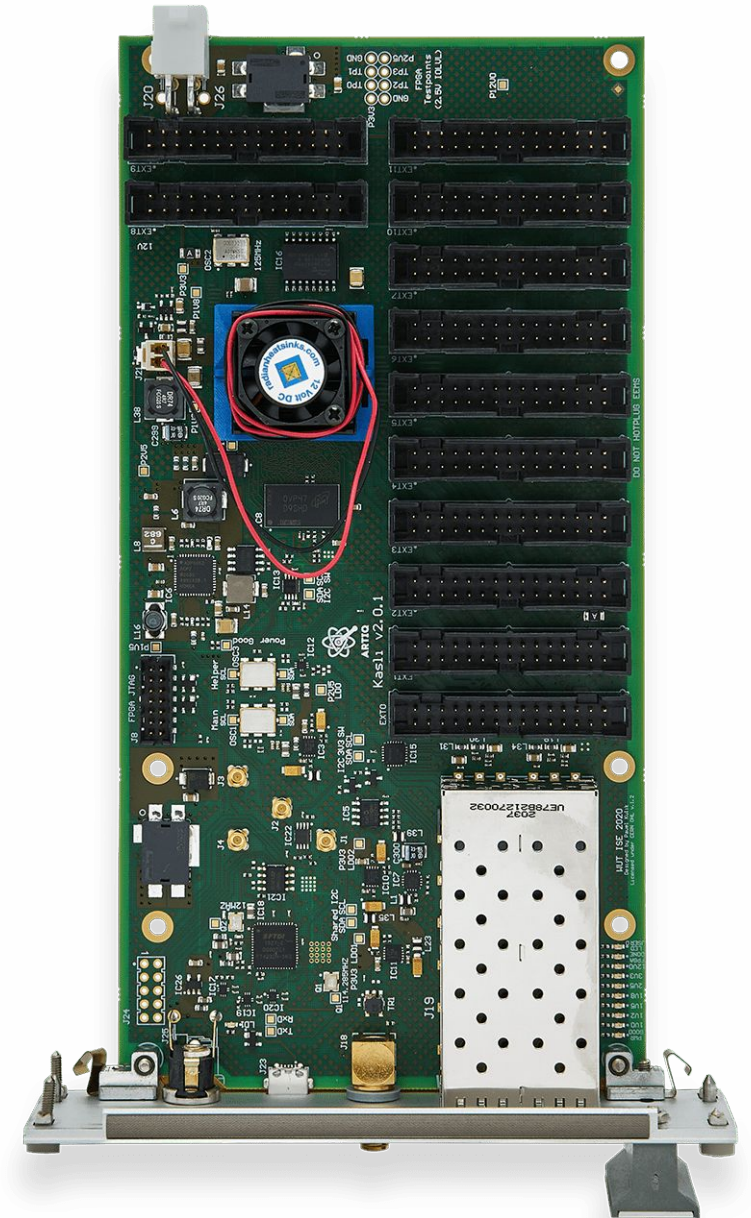
More detailed description: [m-labs.hk/artiq/manual/compiler.html](http://m-labs.hk/artiq/manual/compiler.html)



# 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](http://m-labs.hk/artiq/manual/compiler.html)



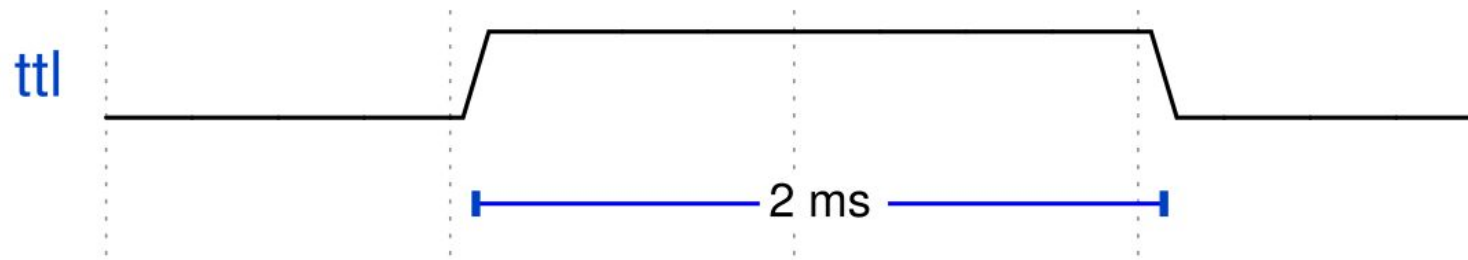
# ARTIQ real-time concepts

```
t1.on()  
delay(2*us)  
t1.off()
```

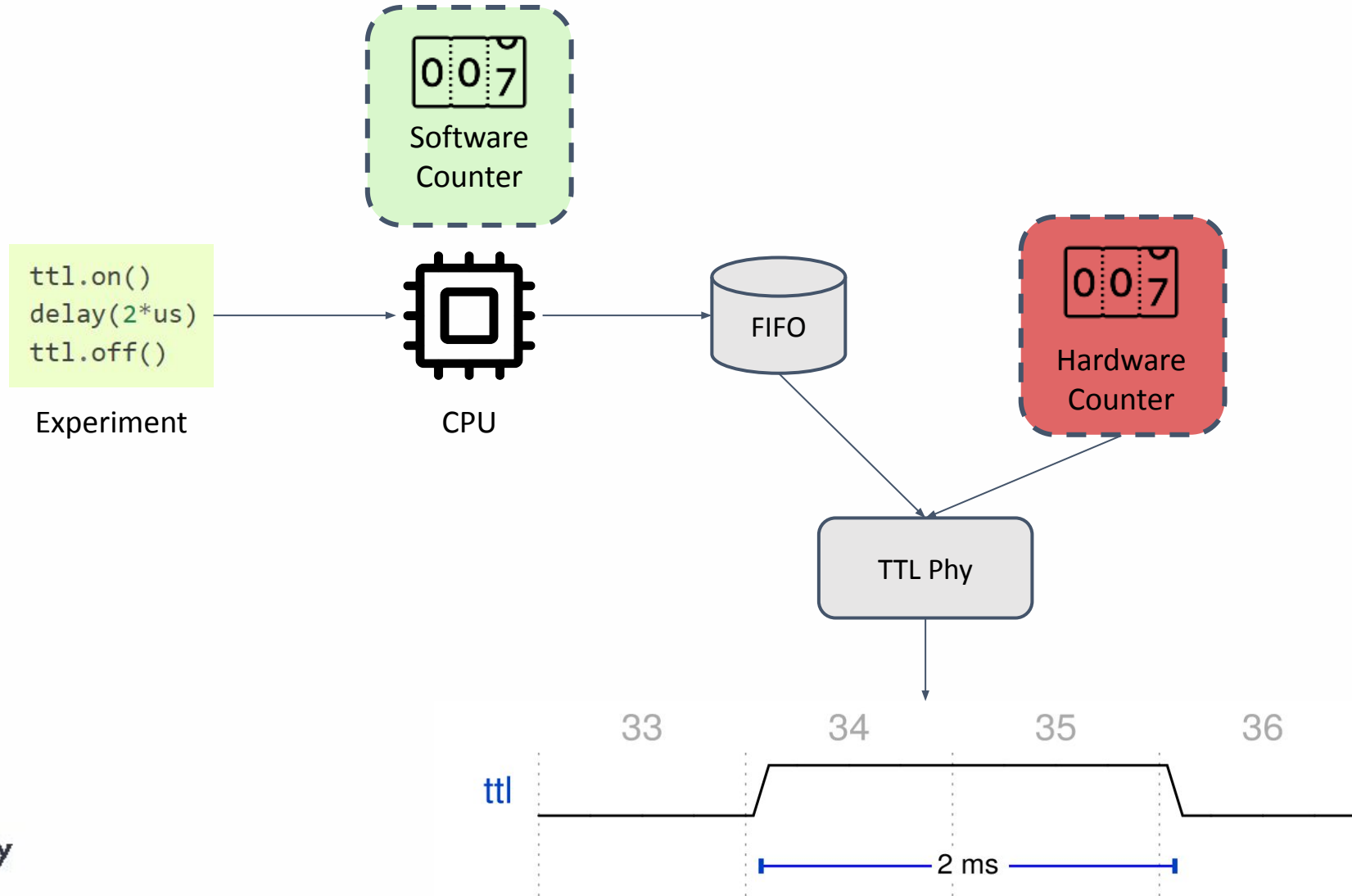


# ARTIQ real-time concepts

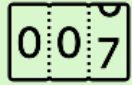
```
t1.on()  
delay(2*us)  
t1.off()
```



# ARTIQ real-time concepts



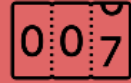
# ARTIQ real-time concepts



Software  
Counter

Timeline cursor  
(now)

Altered by  
experiment's  
commands

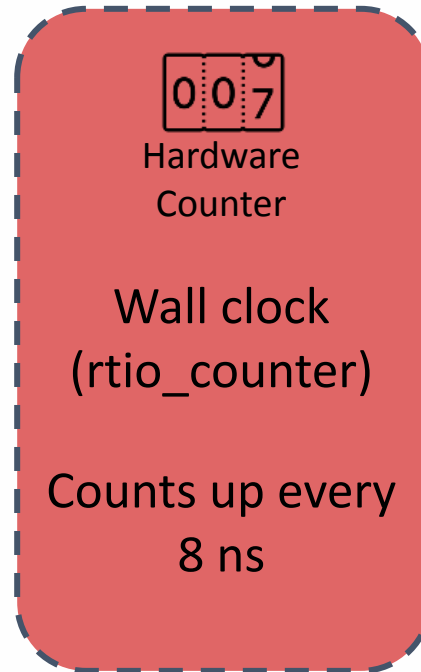
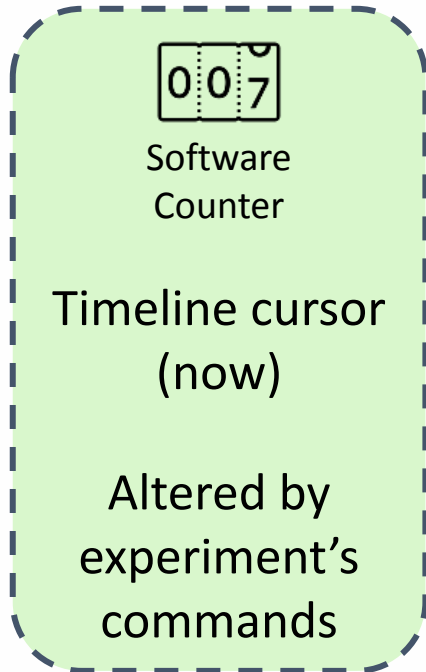


Hardware  
Counter

Wall clock  
(rtio\_counter)

Counts up every  
8 ns

# ARTIQ real-time concepts



**SLACK**

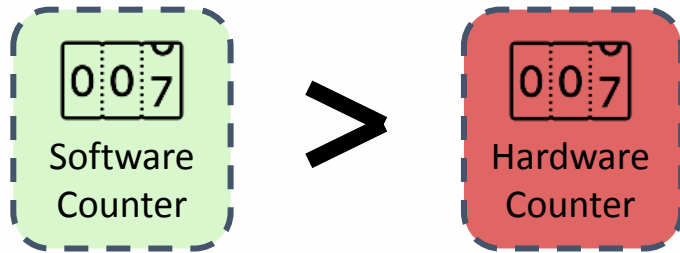


# ARTIQ real-time concepts

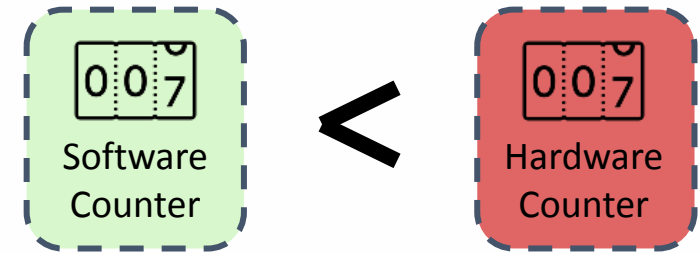
## SLACK

**POSITIVE**

**NEGATIVE**



Planning the future



Altering the past

RTIOUnderflowException

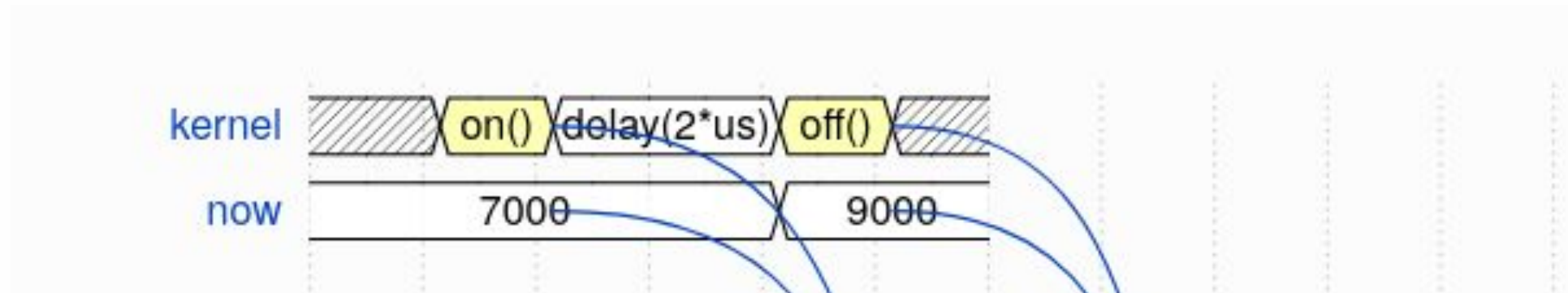
# ARTIQ real-time concepts



ARTIQ experiment execution

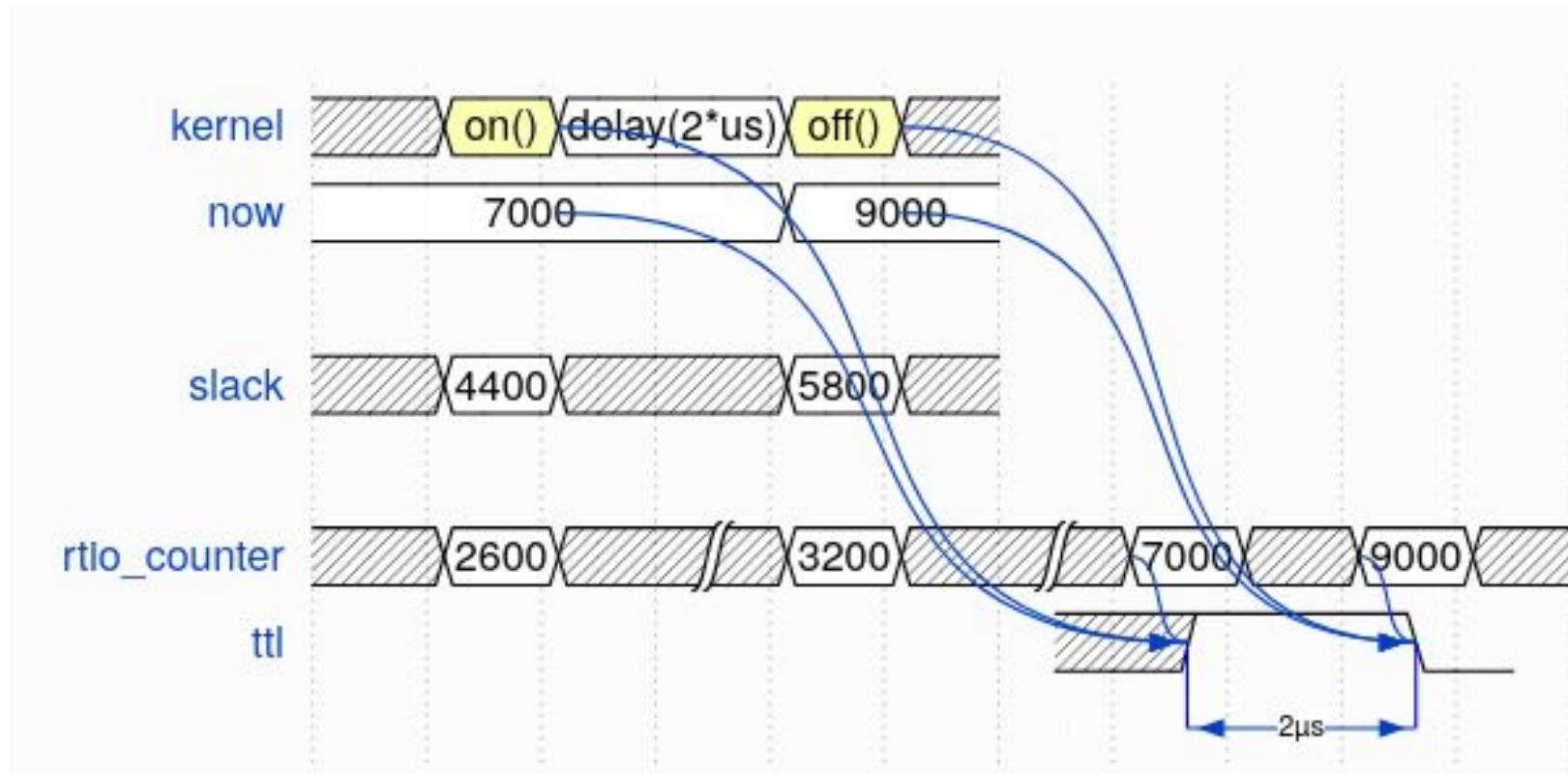
# ARTIQ real-time concepts

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



# ARTIQ real-time concepts

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

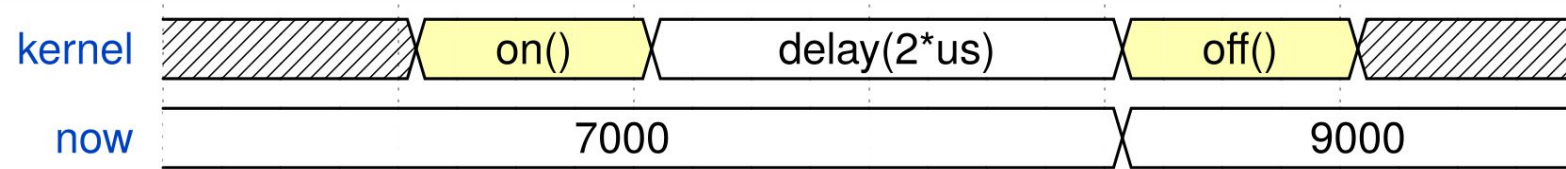


# ARTIQ real-time concepts

## Manipulating timeline

Delay:

`delay(2*us)`



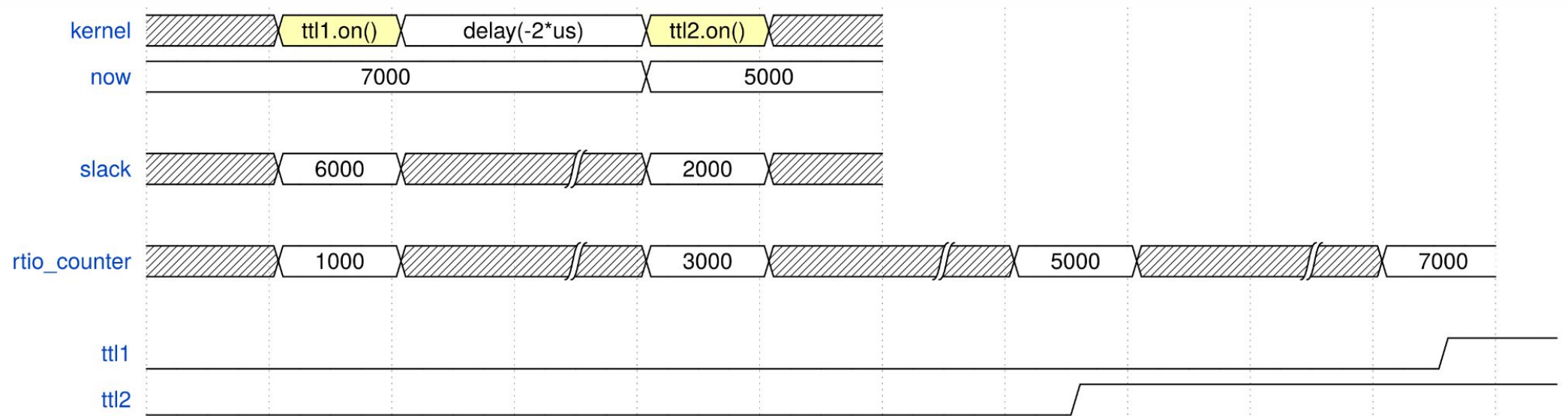


# ARTIQ real-time concepts

## Manipulating timeline

Delay:

`delay(-2*us)`



# ARTIQ real-time concepts

## Manipulating timeline

Current software counter (now):

```
somewhen_mu = now_mu()
```

Seconds to machine units:

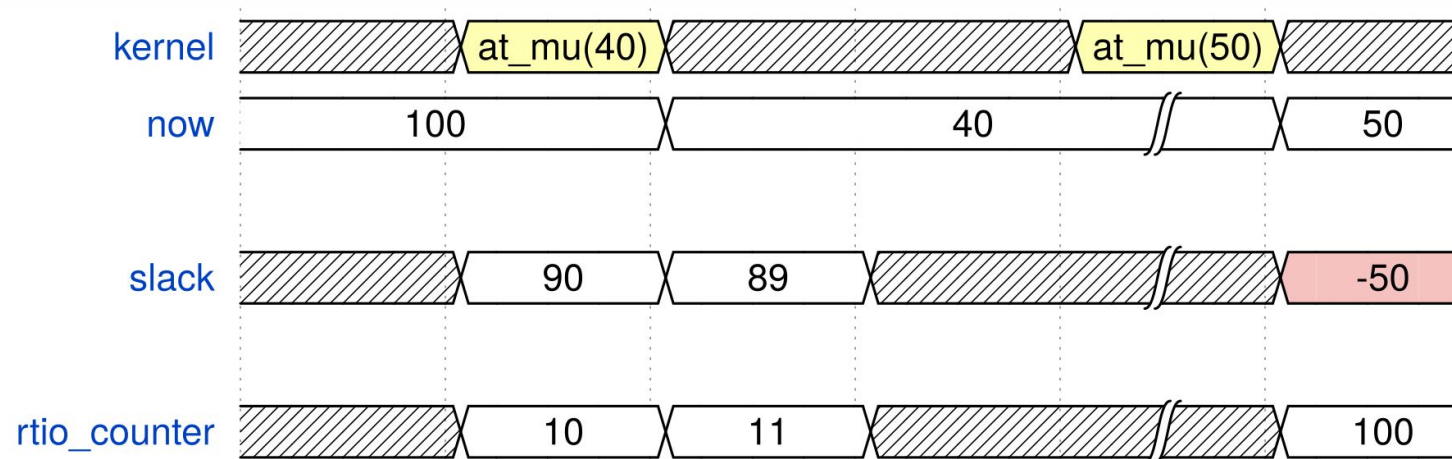
```
time_mu = \  
    self.core.seconds_to_mu(12*us)
```

# ARTIQ real-time concepts

## Manipulating timeline

Setting software counter (*now* cursor):

`at_mu(some_value_mu)`

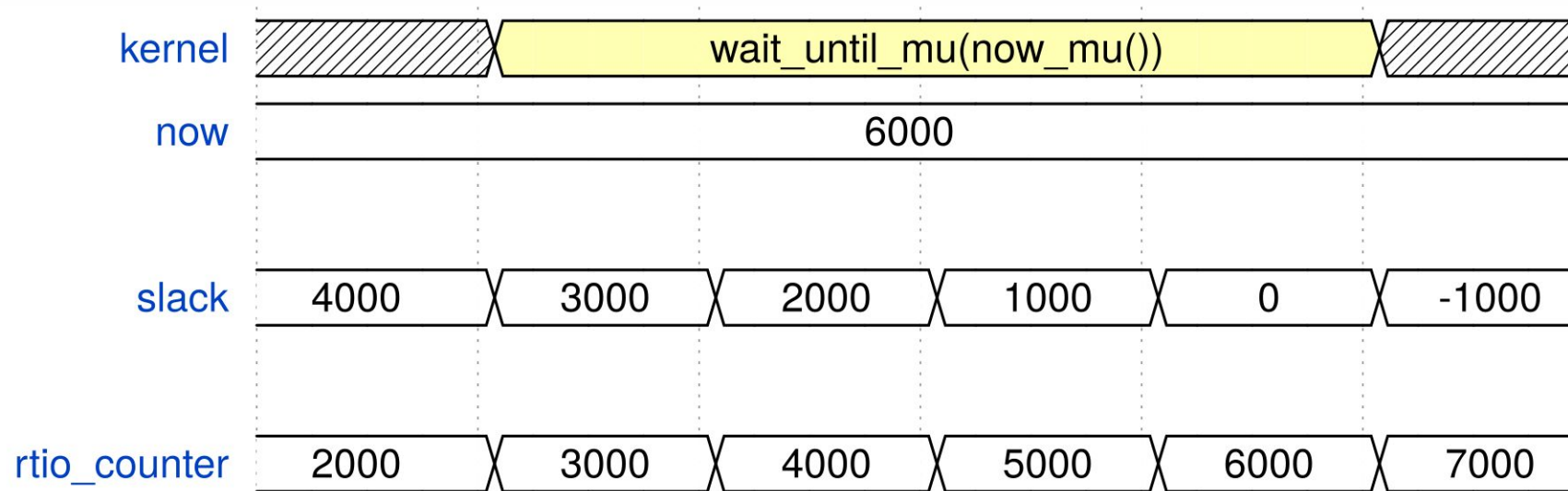


# ARTIQ real-time concepts

## Manipulating timeline

Hardware-software counter synchronization:

```
self.core.wait_until_mu(now_mu())
```

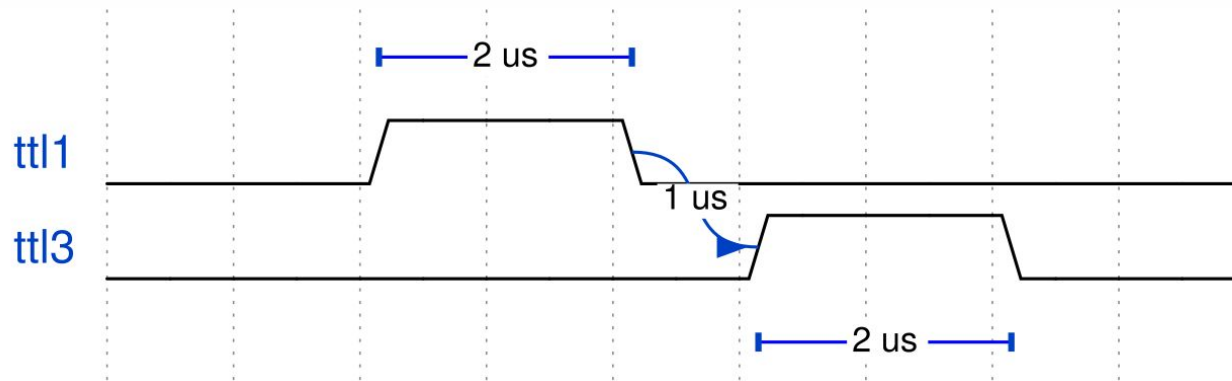


# Timing examples - exercise Timing1

**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.tt11` and `self.tt13`.



## TTL methods:

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

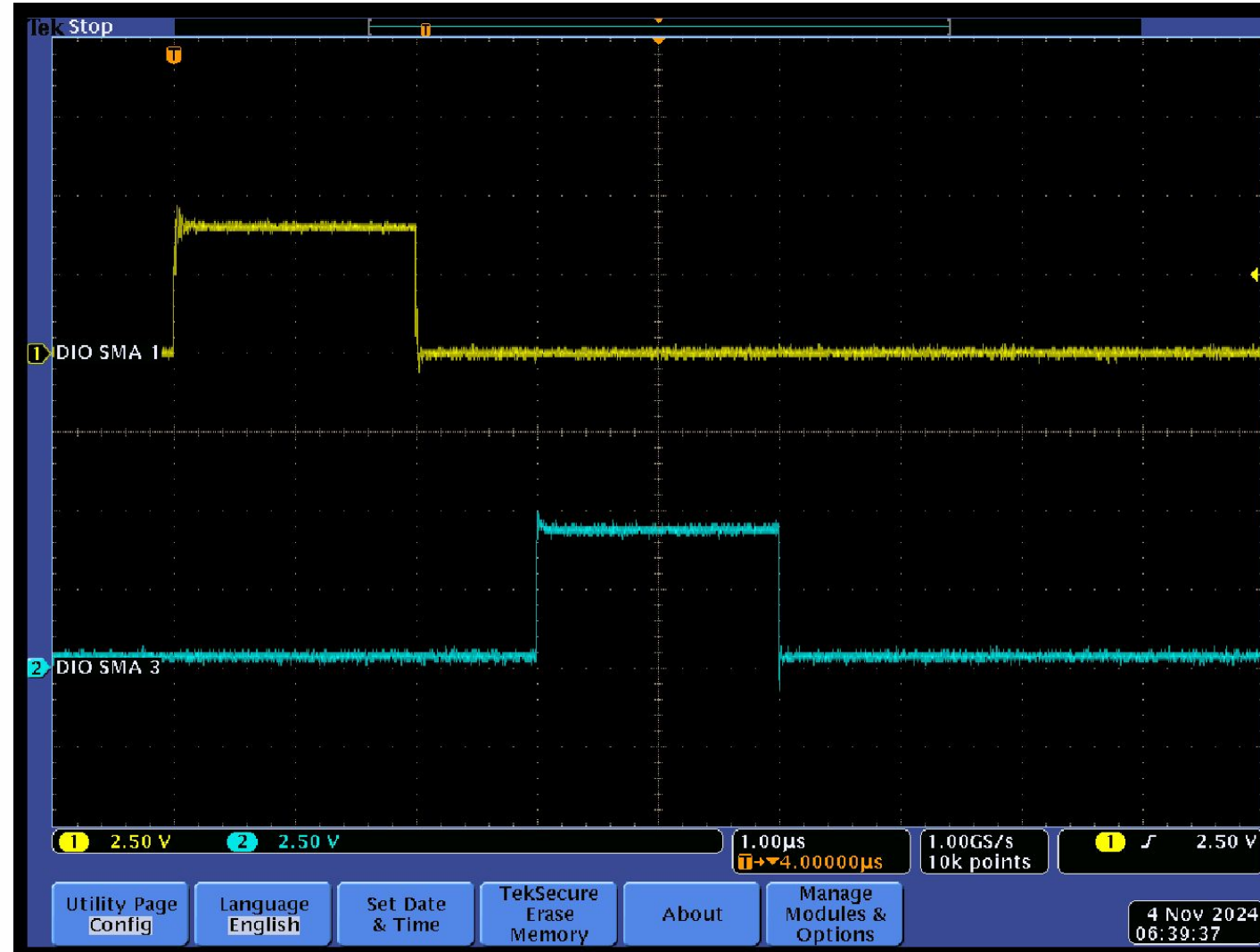
## Timing functions:

- `delay(duration)`



# Timing examples - exercise Timing1

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

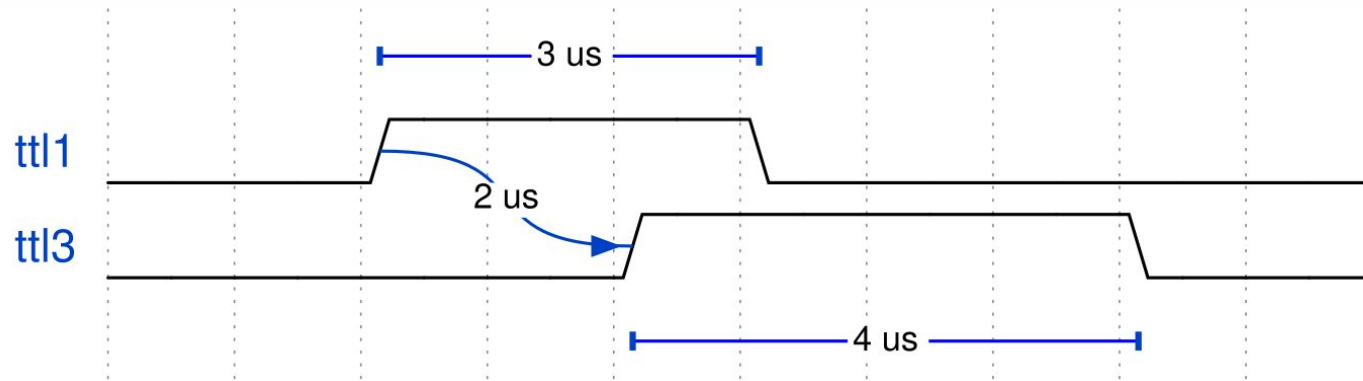


# Timing examples - exercise Timing2

**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.tt11` and `self.tt13`.



## TTL methods:

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

## Timing functions:

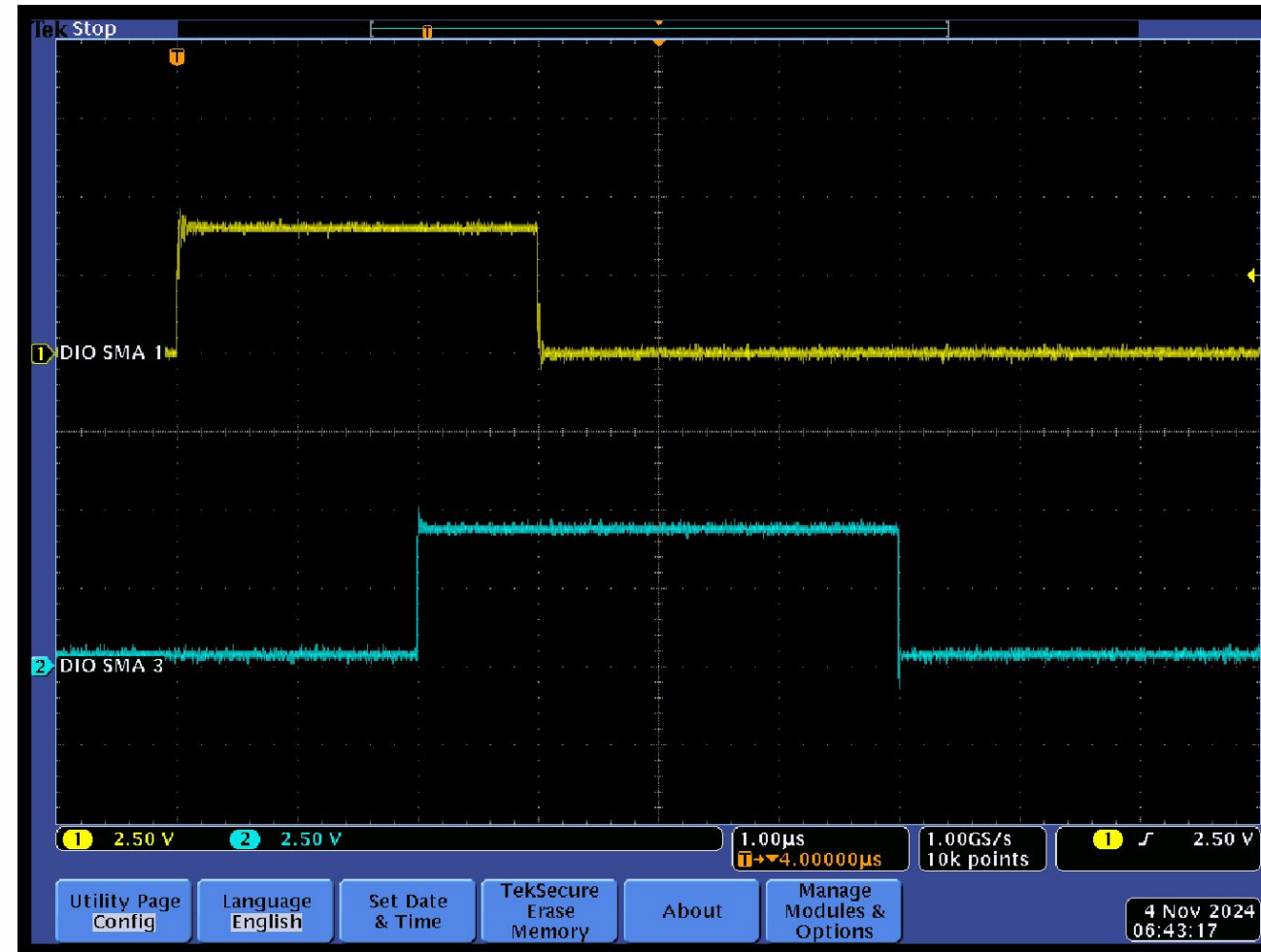
- `now_mu(duration)`
- `at_mu(time_mu)`
- `self.core.seconds_to_mu(time_in_sec)`

# 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.tt11.pulse(3*us)

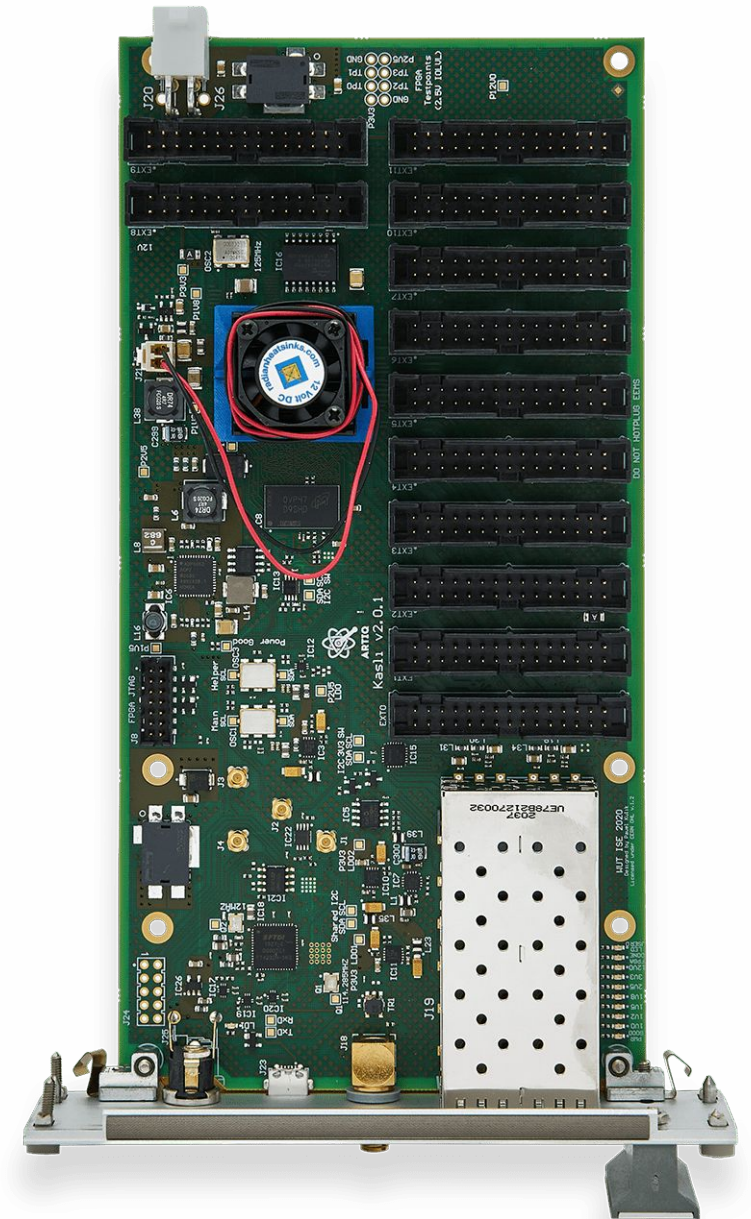
# Let's move counter to the value
# corresponding to the start of the second
# pulse.
at_mu(t + self.core.seconds_to_mu(2*us))
self.tt13.pulse(4*us)
```



# 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](http://m-labs.hk/artiq/manual/compiler.html)

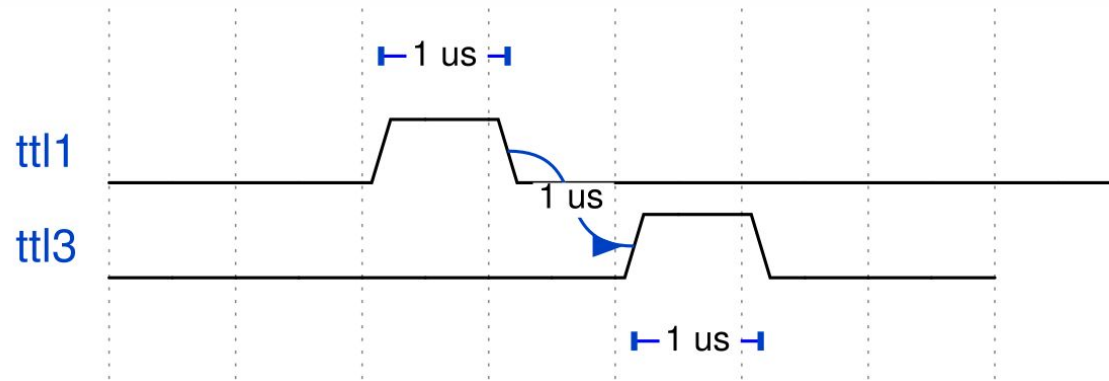


# Timing examples - exercise Timing3

**Exercise:** `timing3.py`

**Solution:** `timing3_solution.py`

**Goal:** Generate 1 us long pulse on `self.tt11` followed by 1 us delay and 1 us long pulse on `self.tt13`. Just after `self.tt11` 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)`

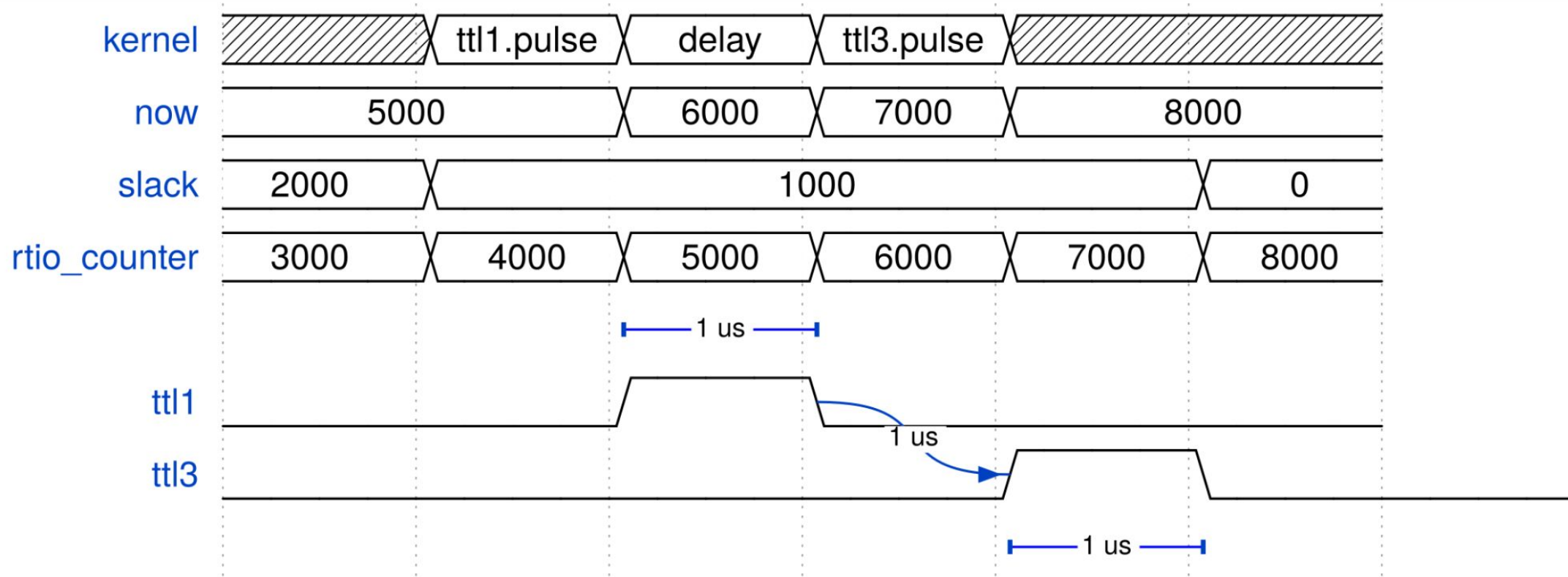


# Timing examples - exercise Timing3

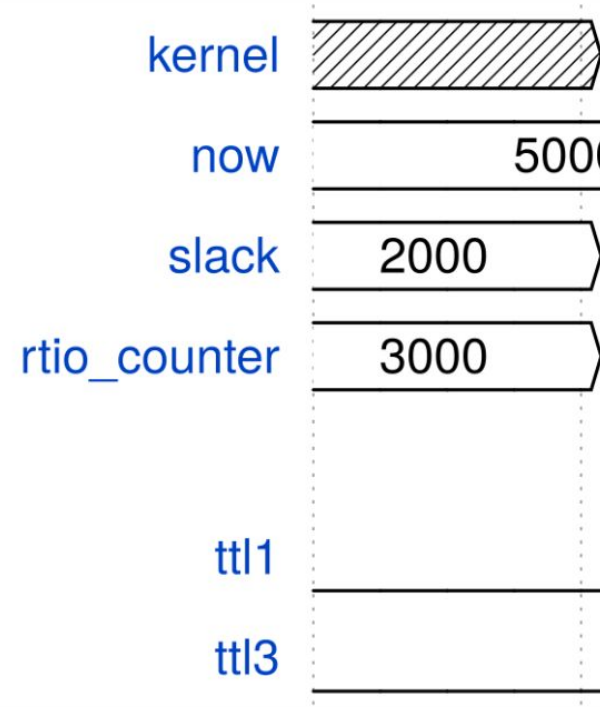
```
self.ttl1.pulse(1*us)
print(now_mu())
delay(1*us)
self.ttl3.pulse(1*us)
```



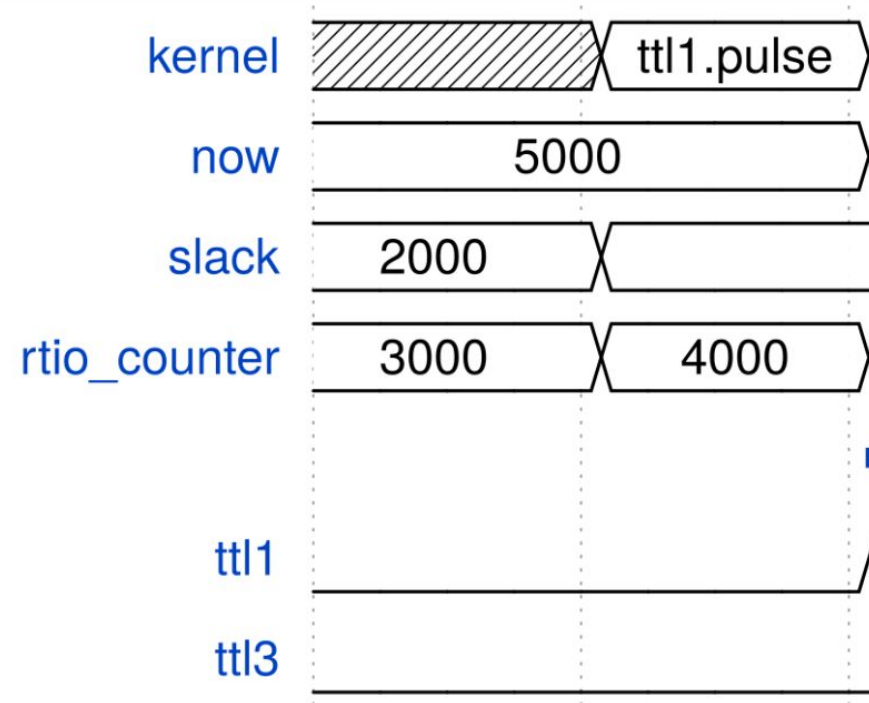
# Timing examples - exercise Timing3 - why exception?



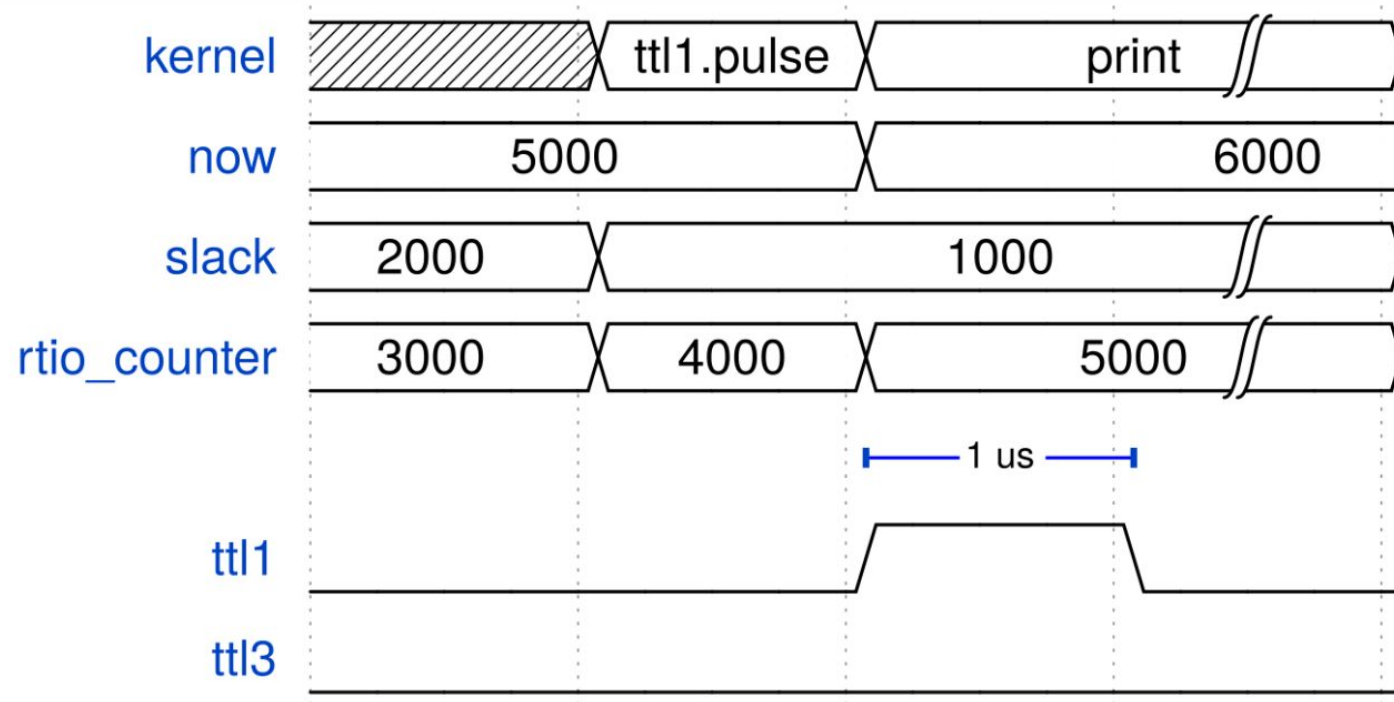
# Timing examples - exercise Timing3 - why exception?



# Timing examples - exercise Timing3 - why exception?

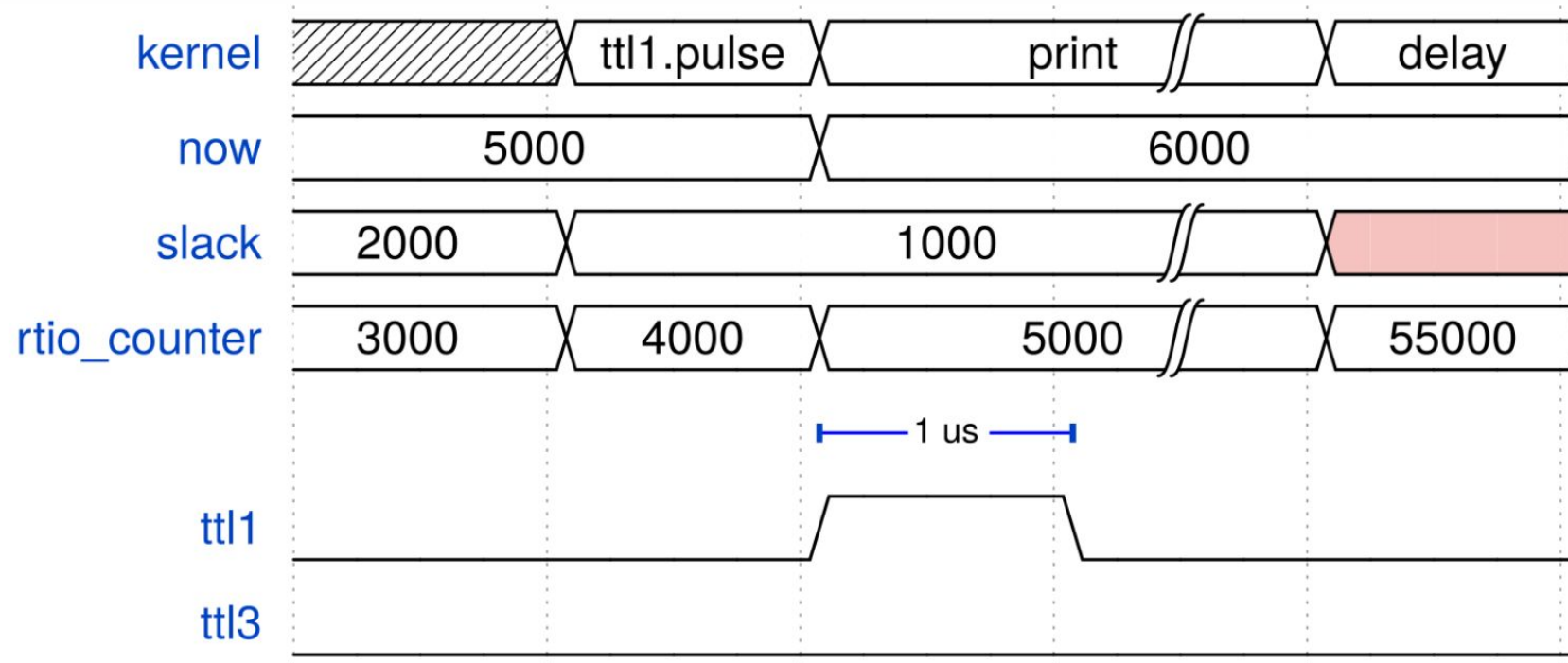


# Timing examples - exercise Timing3 - why exception?

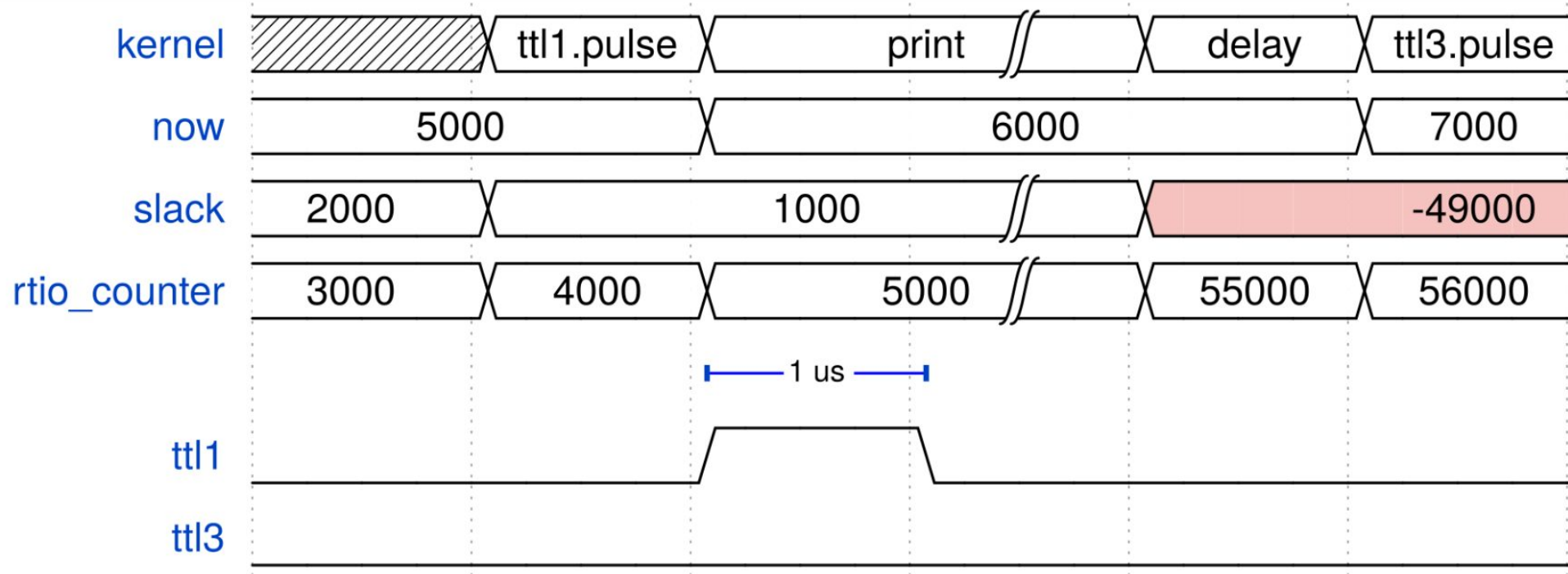




# Timing examples - exercise Timing3 - why exception?



# Timing examples - exercise Timing3 - why exception?

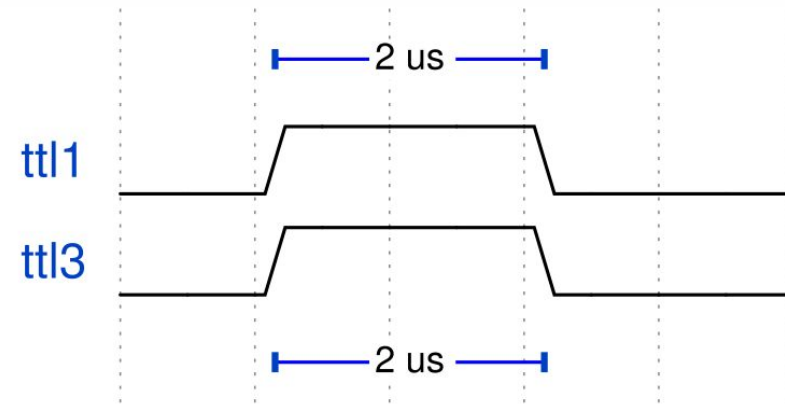


**Altering the past**

**RTIOUnderflowException**

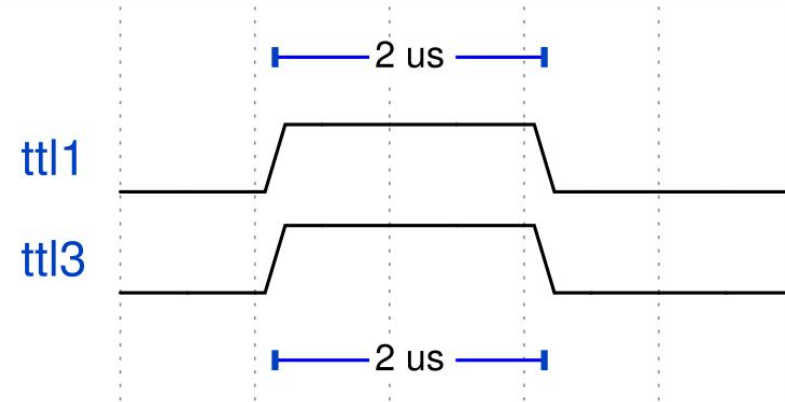
# Parallel and sequential blocks

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

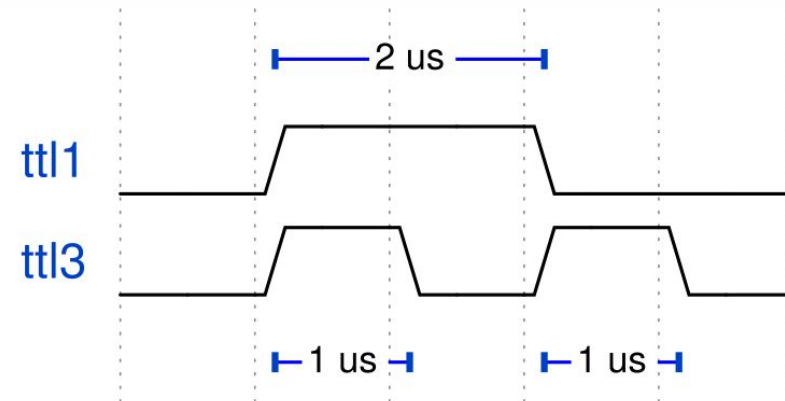


# Parallel and sequential blocks

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

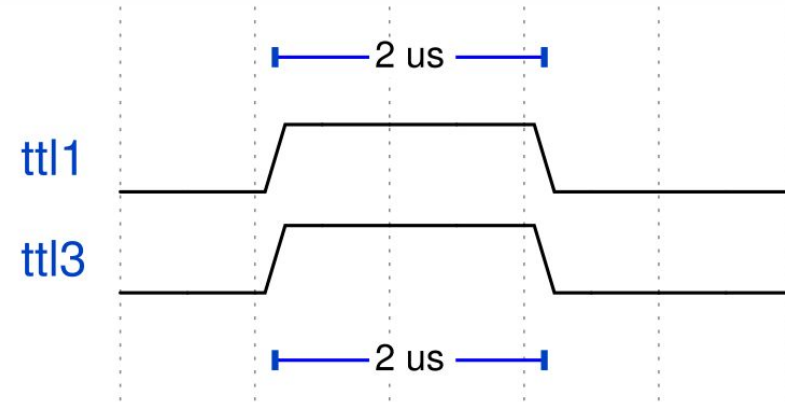


```
with parallel:  
    self.ttl1.pulse(2*us)  
with sequential:  
    self.ttl3.pulse(1*us)  
    delay(1*us)  
    self.ttl3.pulse(1*us)
```

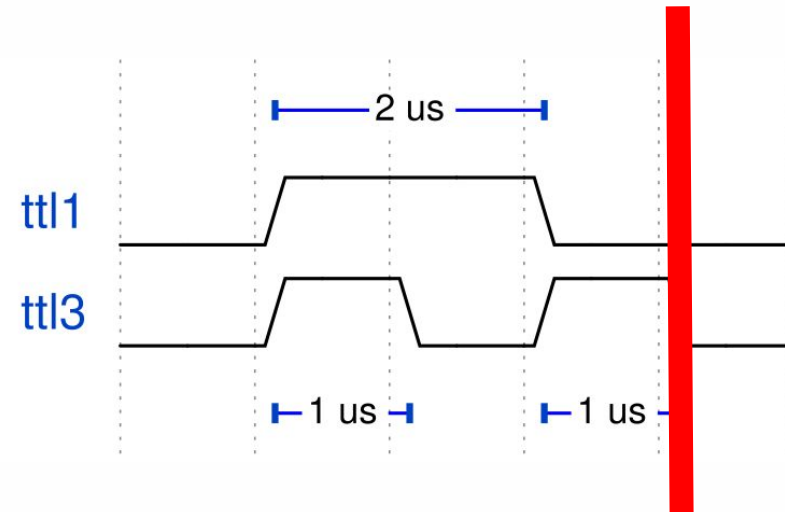


# Parallel and sequential blocks

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

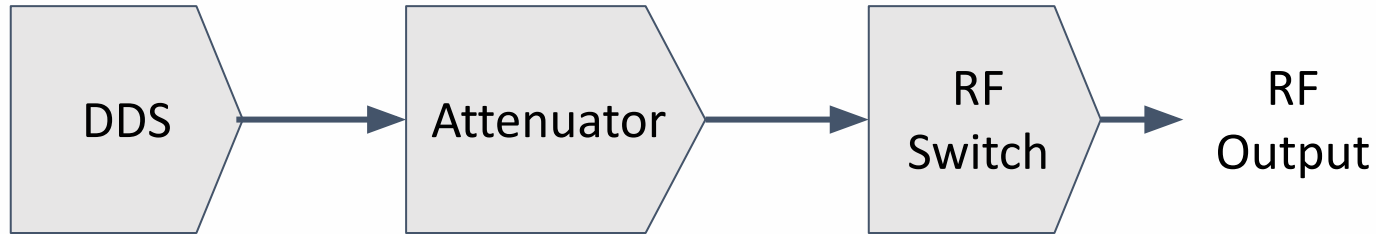


```
with parallel:  
    self.ttl1.pulse(2*us)  
with sequential:  
    self.ttl3.pulse(1*us)  
    delay(1*us)  
    self.ttl3.pulse(1*us)
```



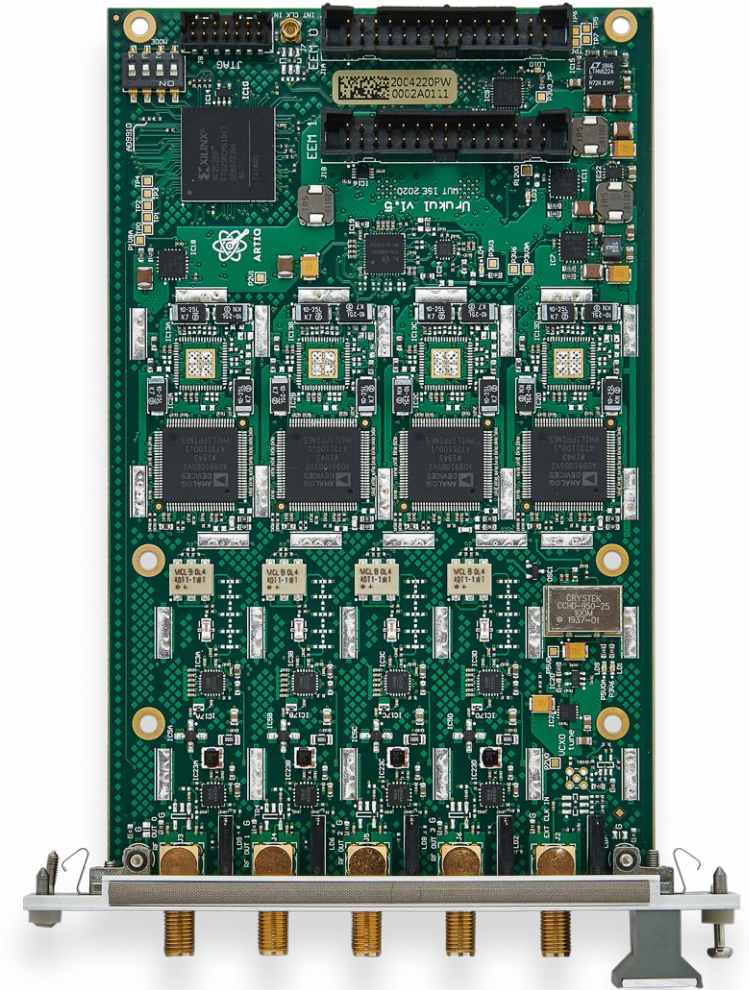


# Controlling Urukul DDS



## 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()*

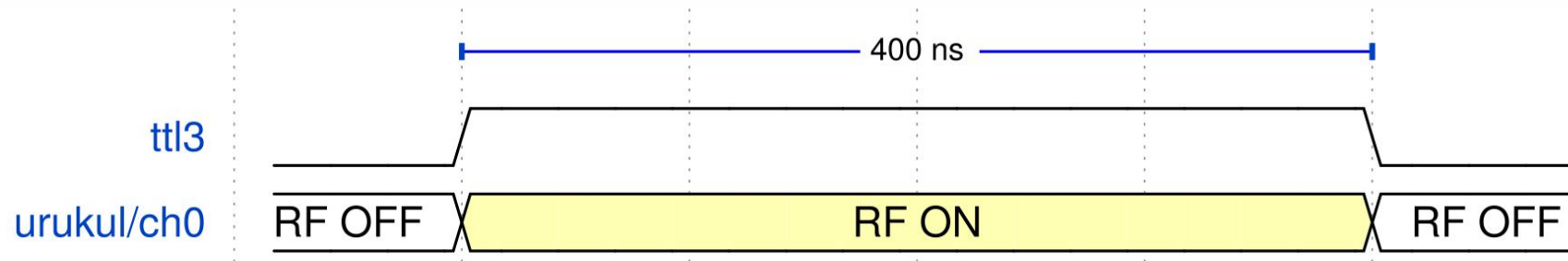


# TTL and Urukul - exercise TTLUrukul1

**Exercise:** `ttl_urukul1.py`

**Solution:** `ttl_urukul1_solution.py`

**Goal:** Generate simultaneous `self.tt13` 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!**

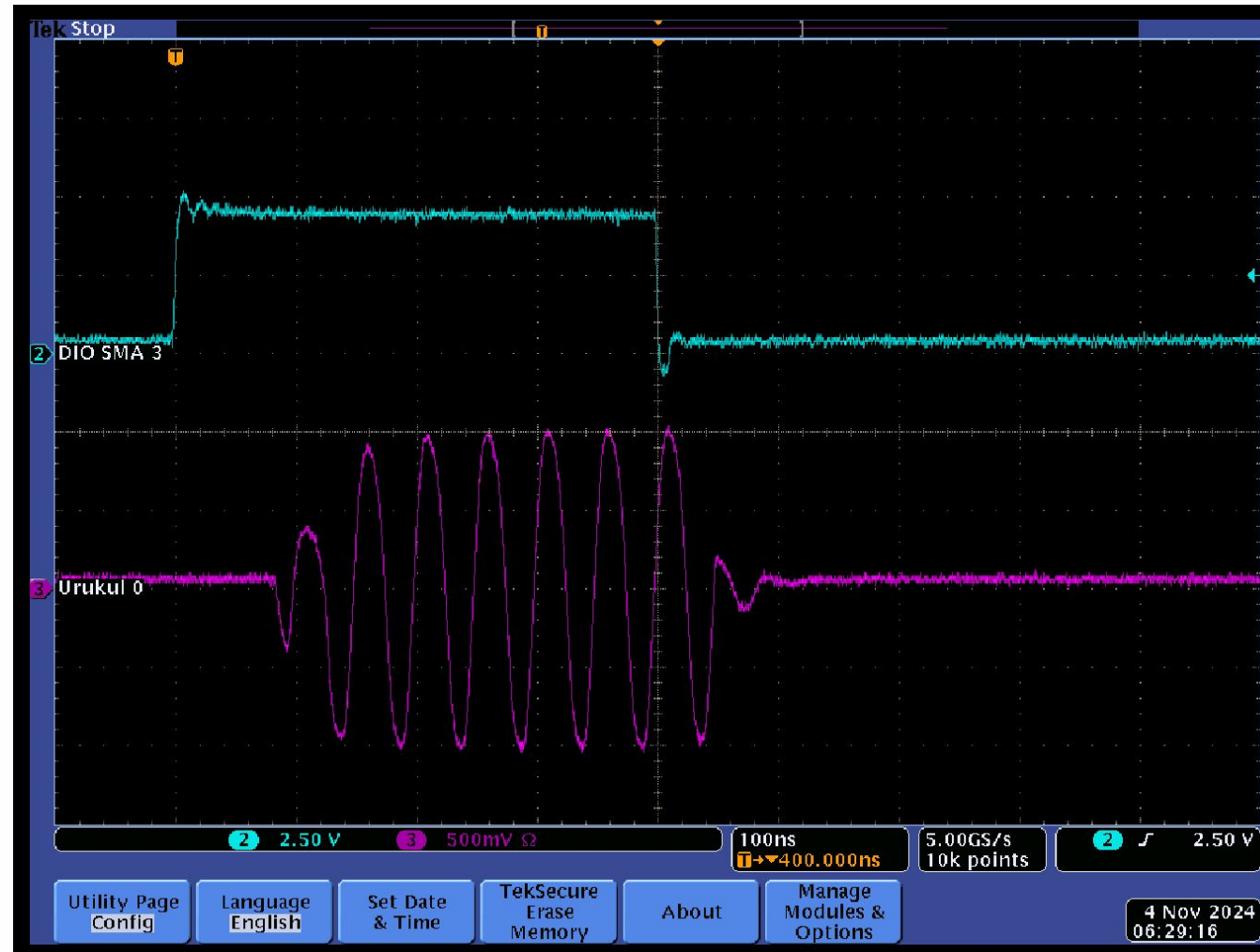
## Urukul channel methods

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

# TTL and Urukul - exercise TTLUrukul1

with parallel:

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

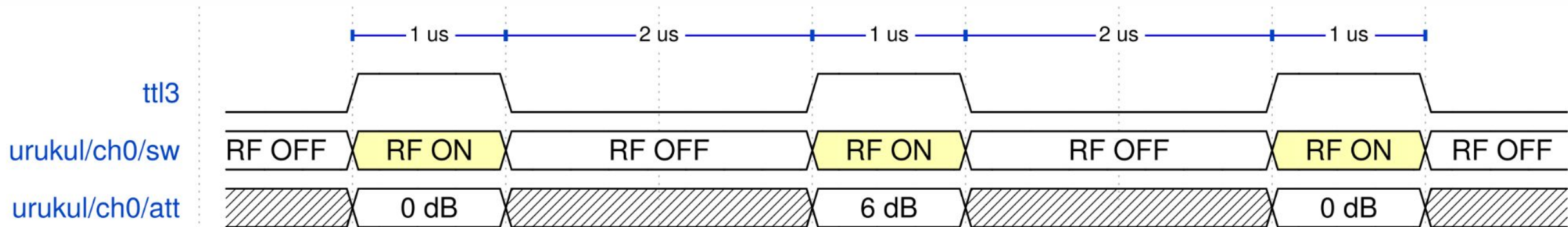


# TTL and Urukul - exercise TTLUrukul2

**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

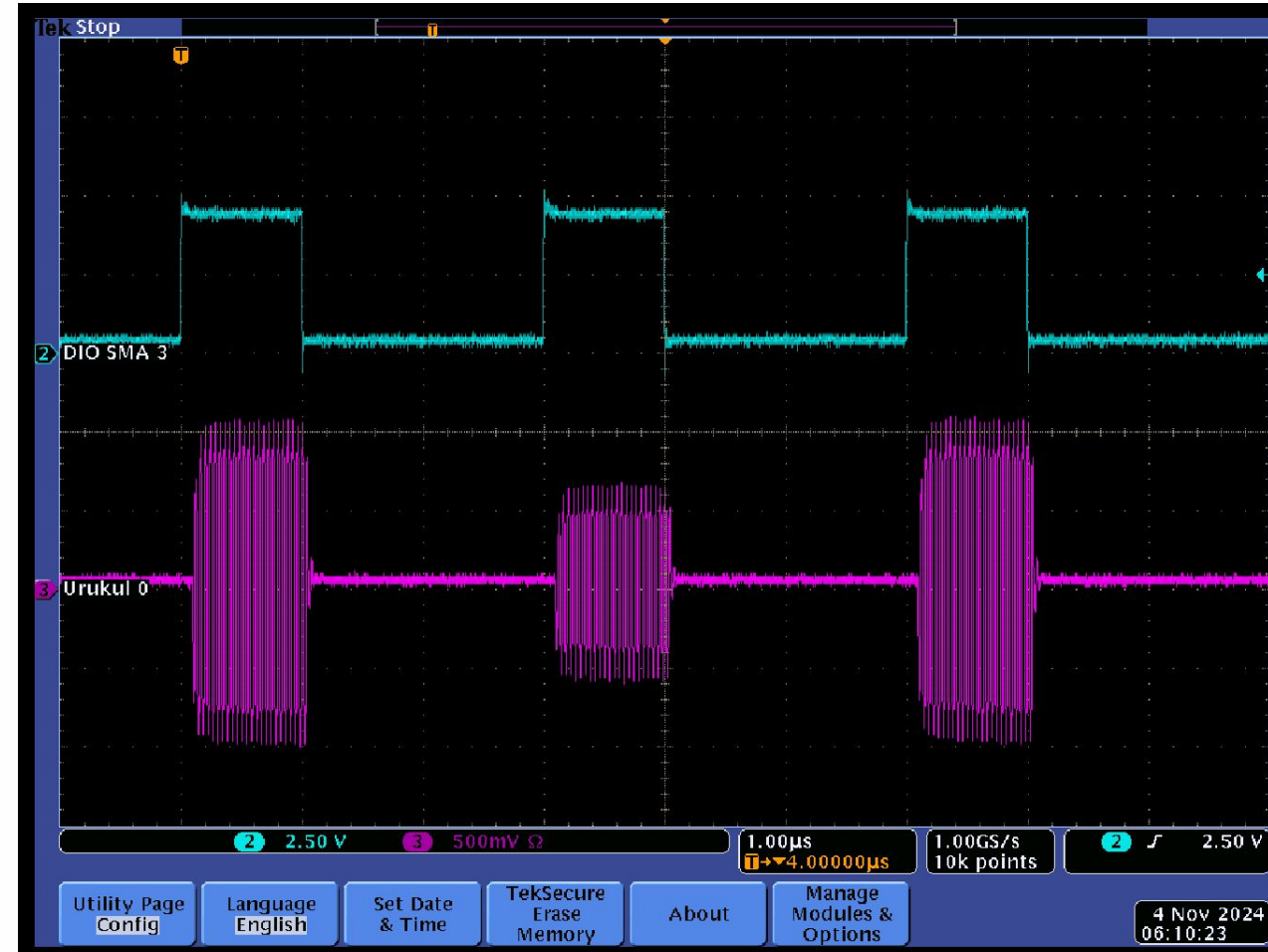
- `channel.set_att(att_dB)`
- `channel.sw` - TTL controlling RF switch  
*TTL output functions apply, i.e. `on()`, `off()`, `pulse()`*

## Timing functions:

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

# 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)
```





# DIO - a closer look at the TTL module

## Connections:

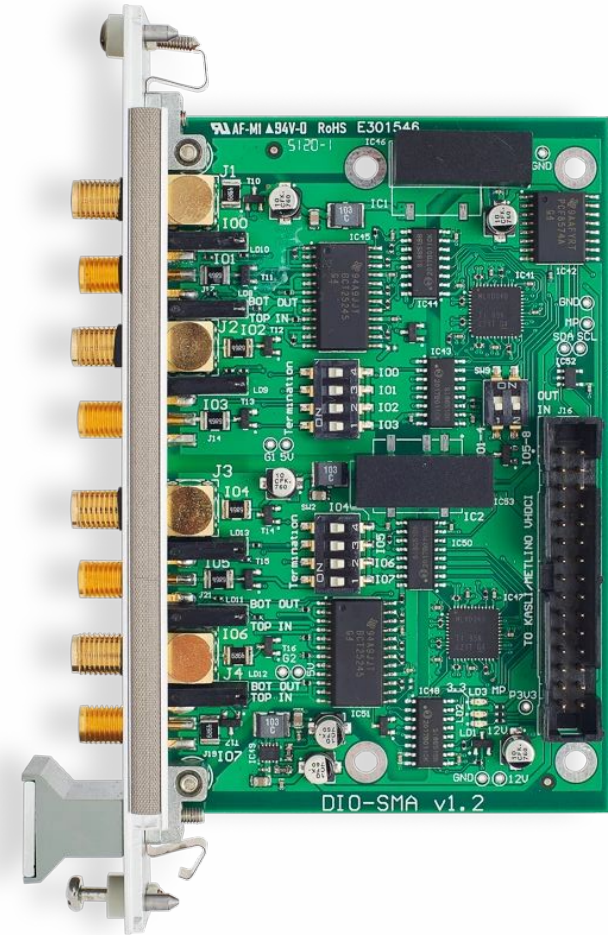
- TTL channel 1 - output to scope CH1 and TTL channel 5
- TTL channel 3 - output to scope CH2
- TTL channel 5 - input with signal fed from channel 1

## 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(timestamp_mu)`

## Optional:

- controller may be equipped with with gateway edge counter





# 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)
```

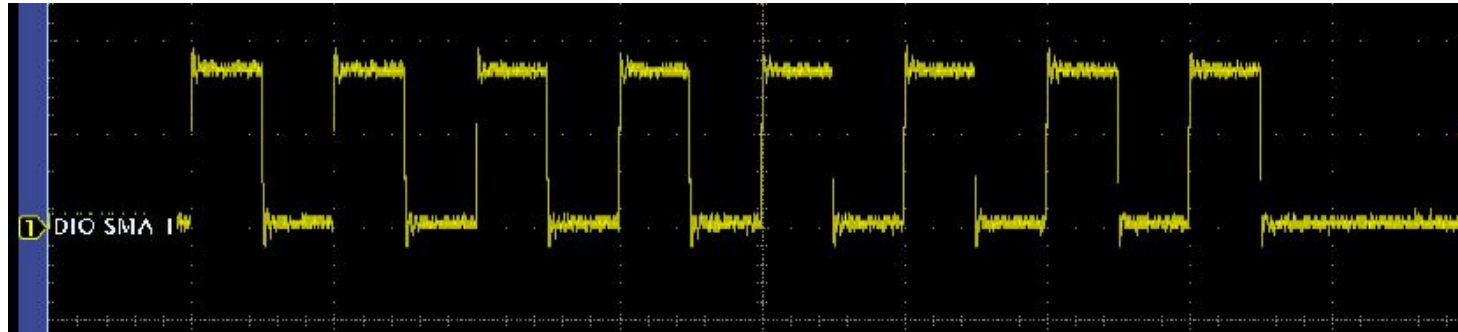
```
for _ in range(5):  
    self.ttl5.sample_get()
```

# TTL examples - exercise TTLSample

**Exercise:** `t11_sample.py`

**Solution:** `t11_sample_solution.py`

**Goal:** There is a square wave signal generated on TTL channel `self.t11` that lasts for 8 us and has period of 1us. Sample signal fed to `self.t15` 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]`.



## 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
```

# TTL examples - exercise TTLGatedInput

**Exercise:** `ttl_gated_input.py`

**Solution:** `ttl_gated_input_solution.py`

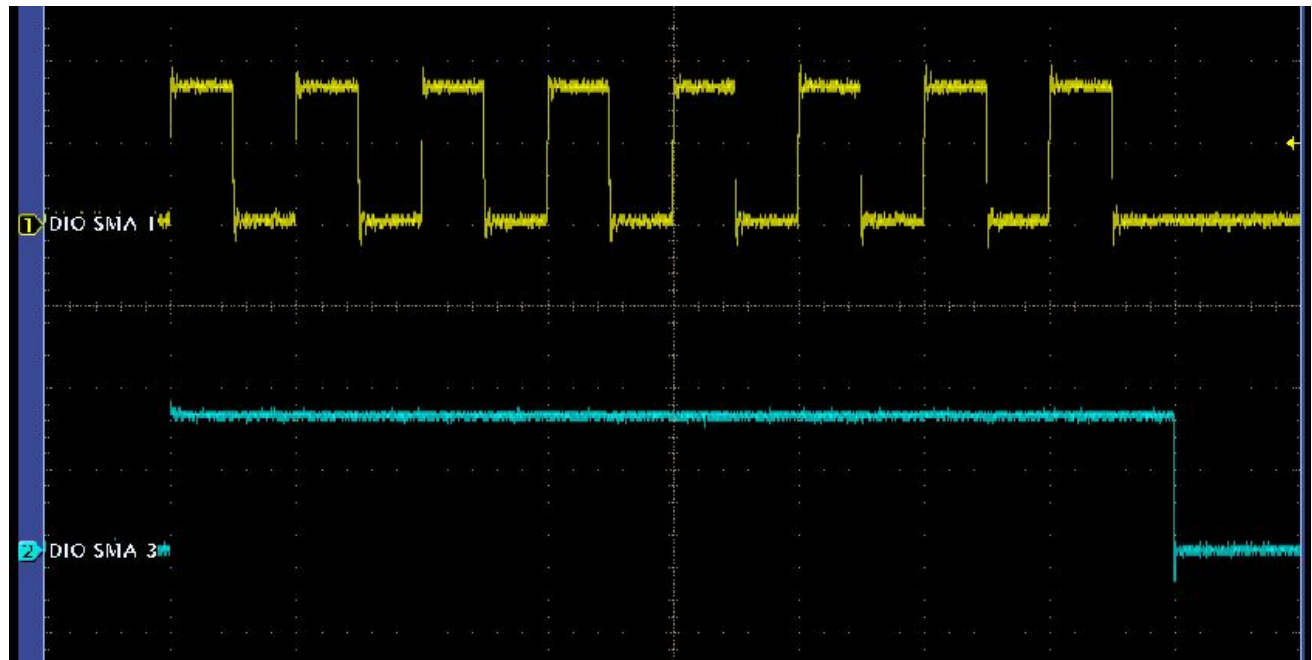
**Goal:** There is a square wave signal generated on TTL channel `self.tt11` that lasts for 8 us and has period of 1us. Count both rising and falling edges of signal fed from `self.tt11` to `self.tt15`. Use `self.tt13` output channel as an indicator of when the gate is open. Print the the number of rising 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)`



# TTL examples - exercise TTLGatedInput cont.

**Exercise:** `ttl_gated_input.py`

**Solution:** `ttl_gated_input_solution.py`

**Goal:** Find out when `RTIOOverflow` exceptions occurs.

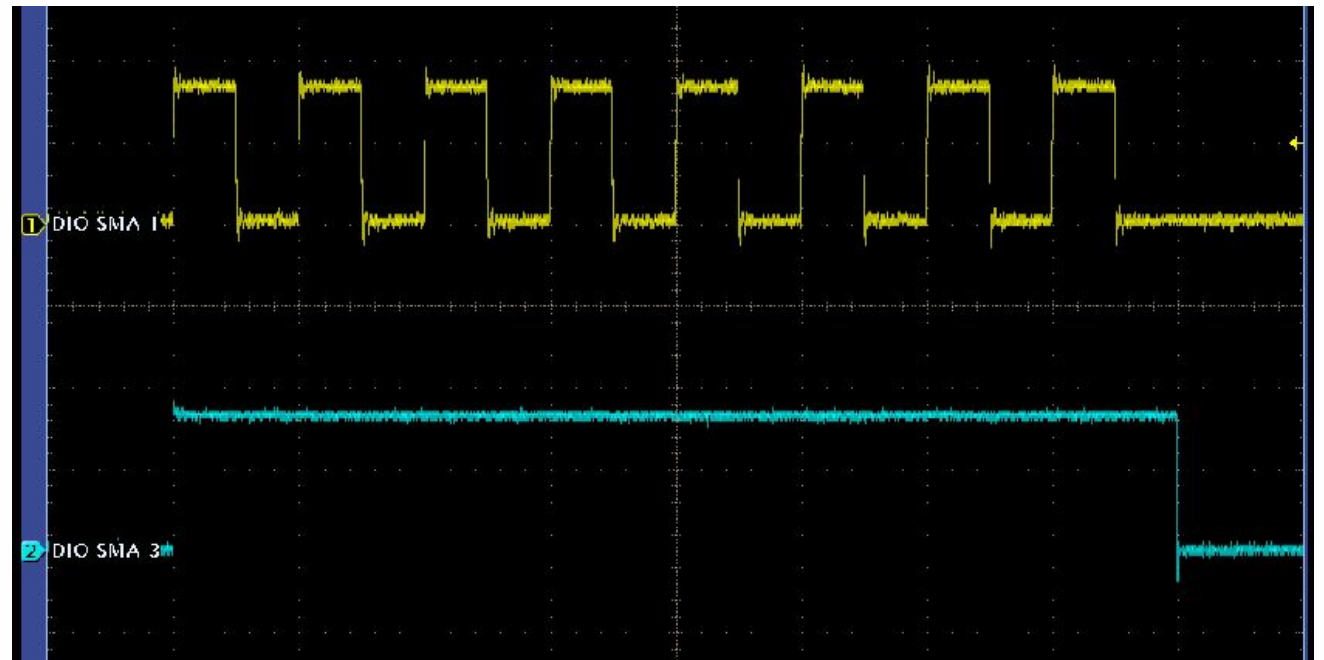
Hint: change the number of events that are to be recorded within the gating period

## 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)`



# 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:** `t11_gated_timestamp.py`

**Solution:** `t11_gated_timestamp_solution.py`

**Goal:** There is a square wave signal generated on TTL channel `self.tt11` that lasts for 8 us and has period of 1us. Open gate window for both rising and falling edges of signal fed from `self.tt11` to `self.tt15`, 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:** `t11_edge_counter.py`

**Solution:** `t11_edge_counter_solution.py`

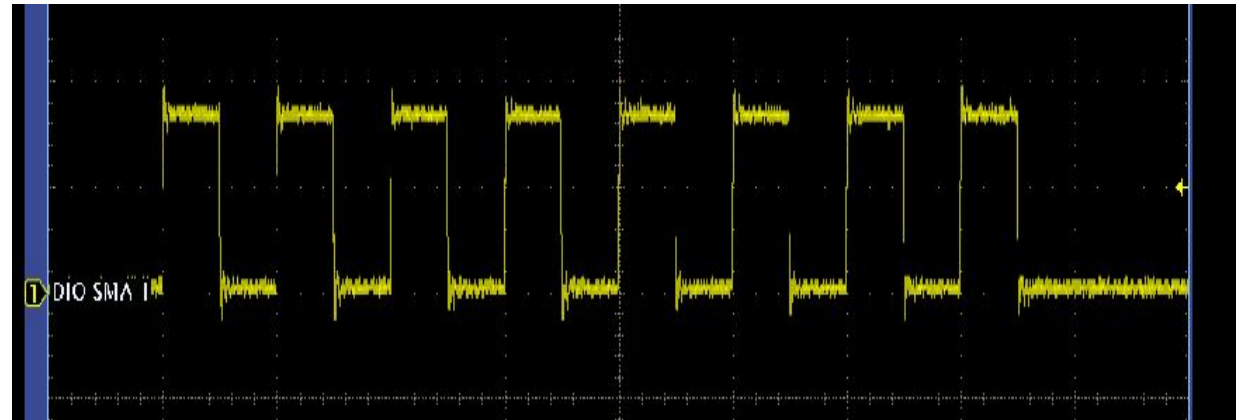
**Goal:** Write missing part of the experiment that generates square-like signal `self.t11` that lasts 60 us and has period of 1 us. This signal is fed by wire to `self.t15`. Count both rising and falling edges of using `self.t11_edge_counter` device. Print the number of rising and falling edges. You 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()`



# Devices that will be used

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

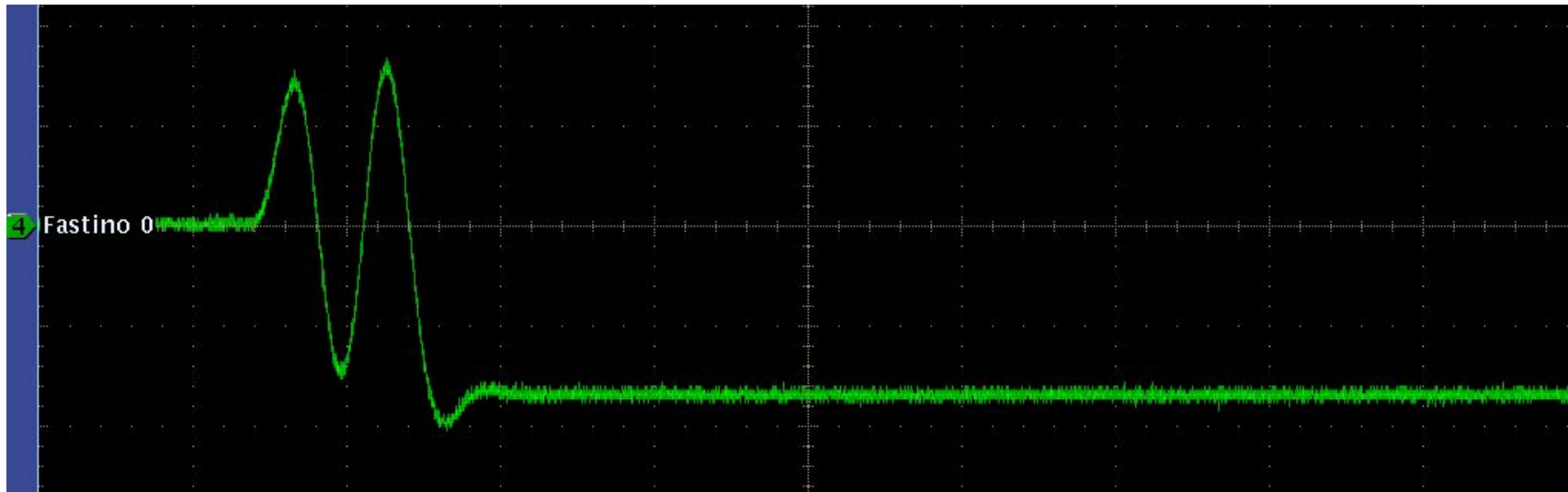


# Basic Fastino (DAC) output - exercise FastinoBasic

**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:**

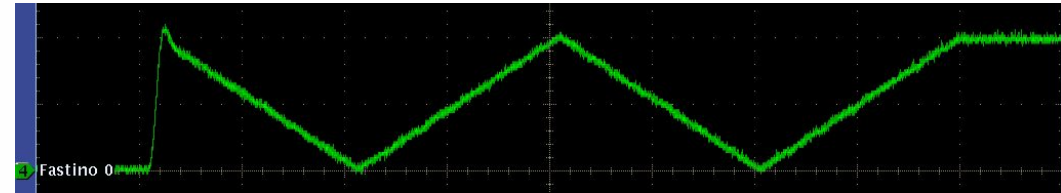
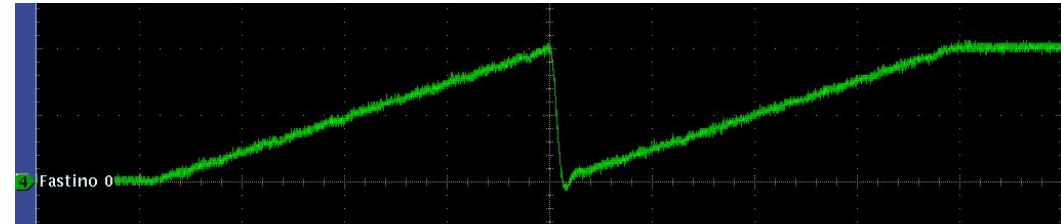
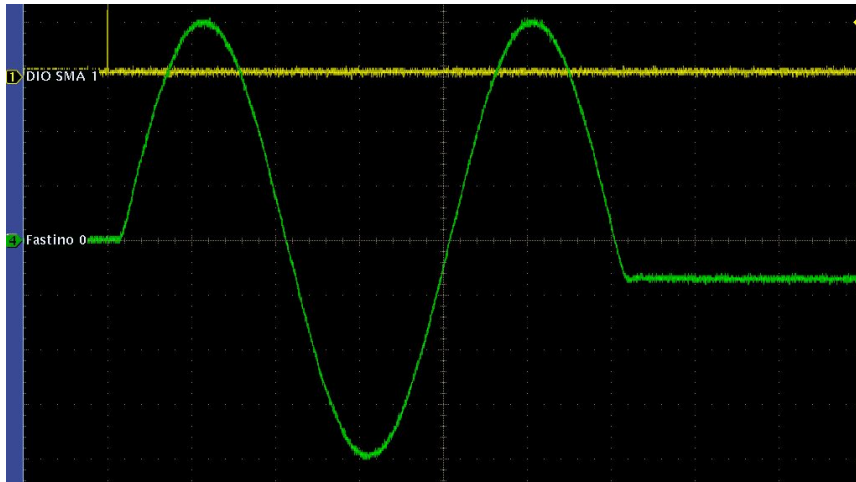
- `delay(duration)`

# 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?



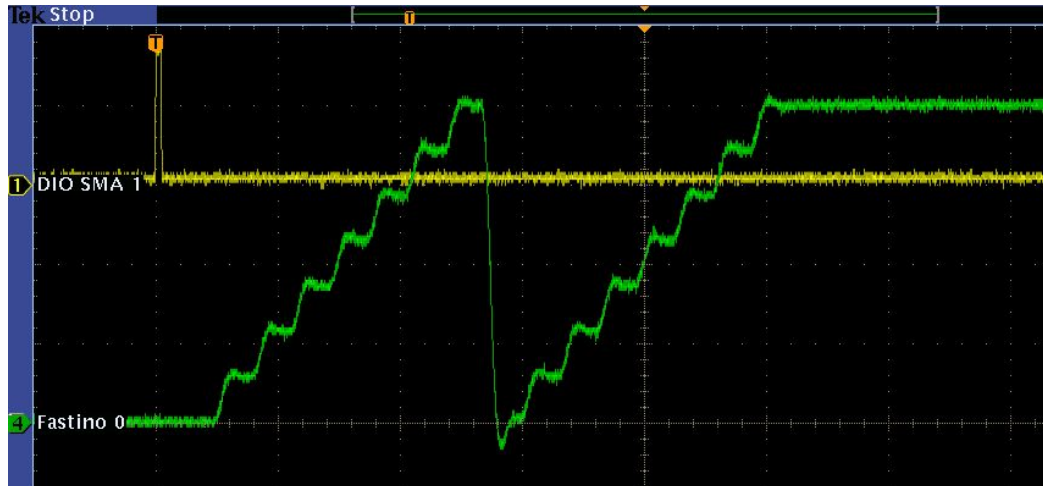
## Methods:

- 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.



Function	Sawtooth	↻
Amplitude	2 V	↕ ↻
Sample_number	16	↕ ↻
Enable_interpolation	<input type="checkbox"/>	↻
Interpolation_rate	8	↕ ↻
Delay_multiplier	8	↕ ↻
Scope_horizontal_scale	10 us	↕ ↻

## Notes:

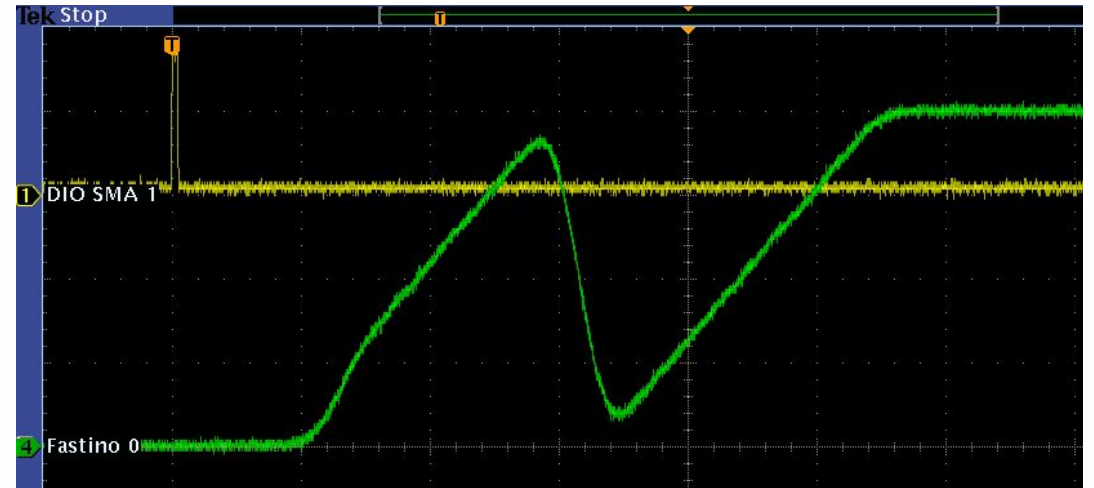
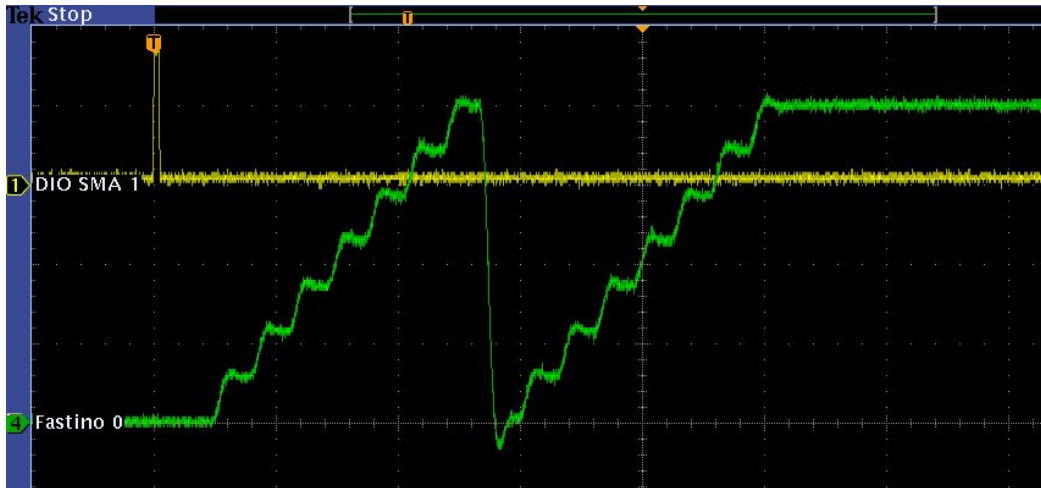
- You may copy your functions from previous exercise to `fastino_interpolation.py`.
- When interpolation is enabled Fastino will only accept one input sample per input sample period.



# 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.



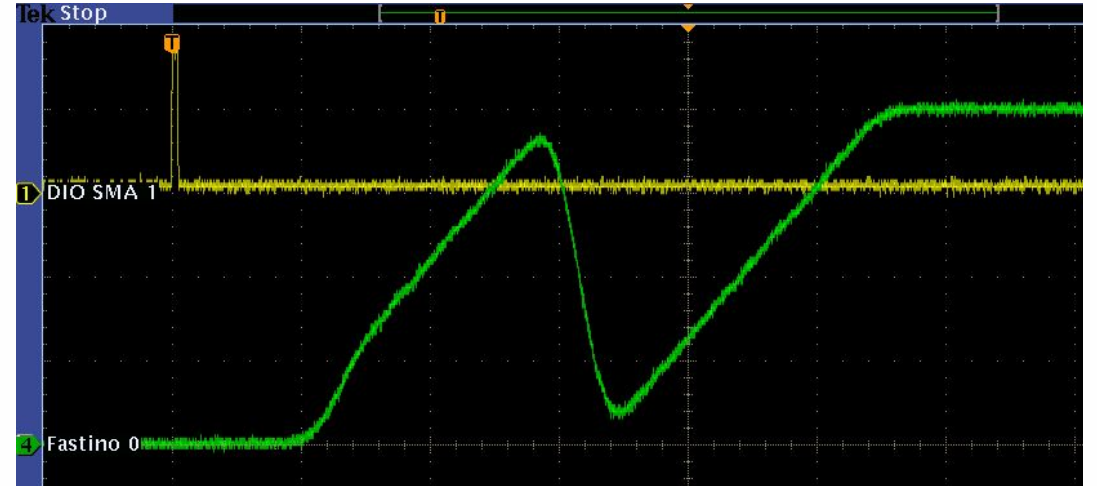
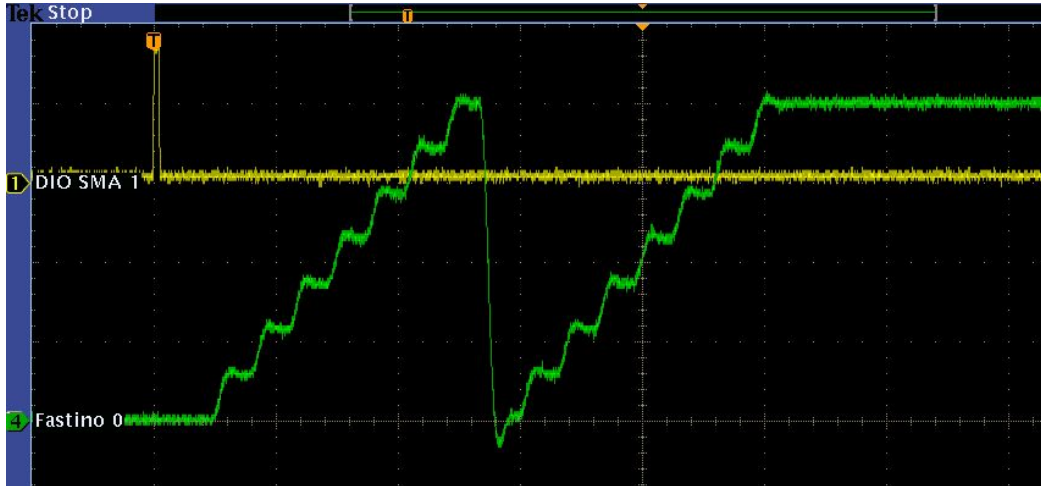
## Notes:

- 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.

# 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.

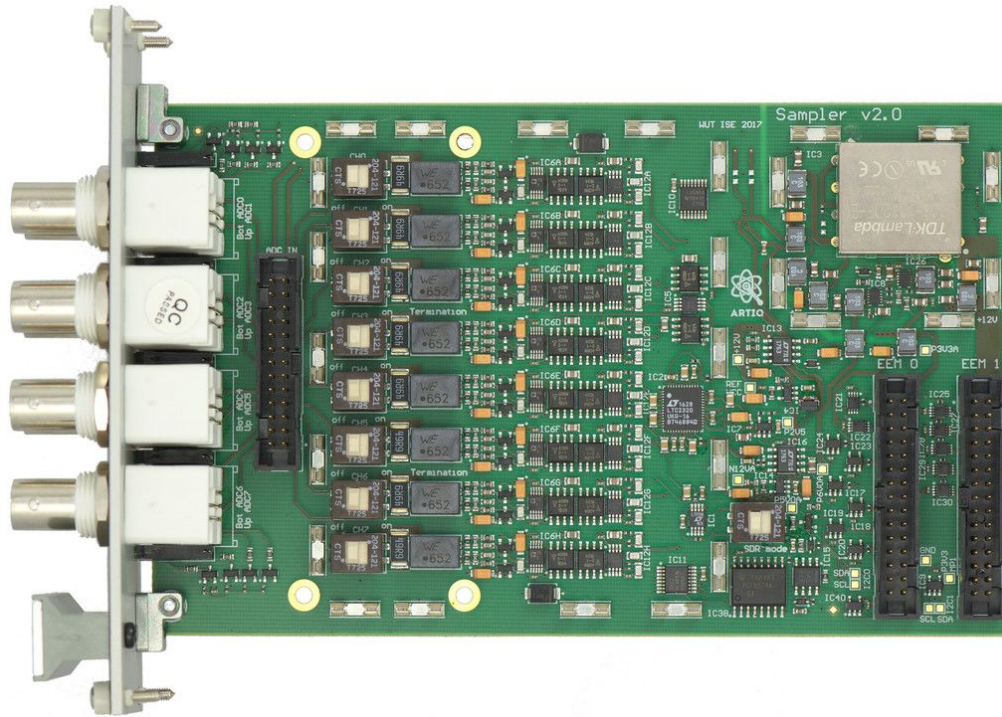


## Notes:

- 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.

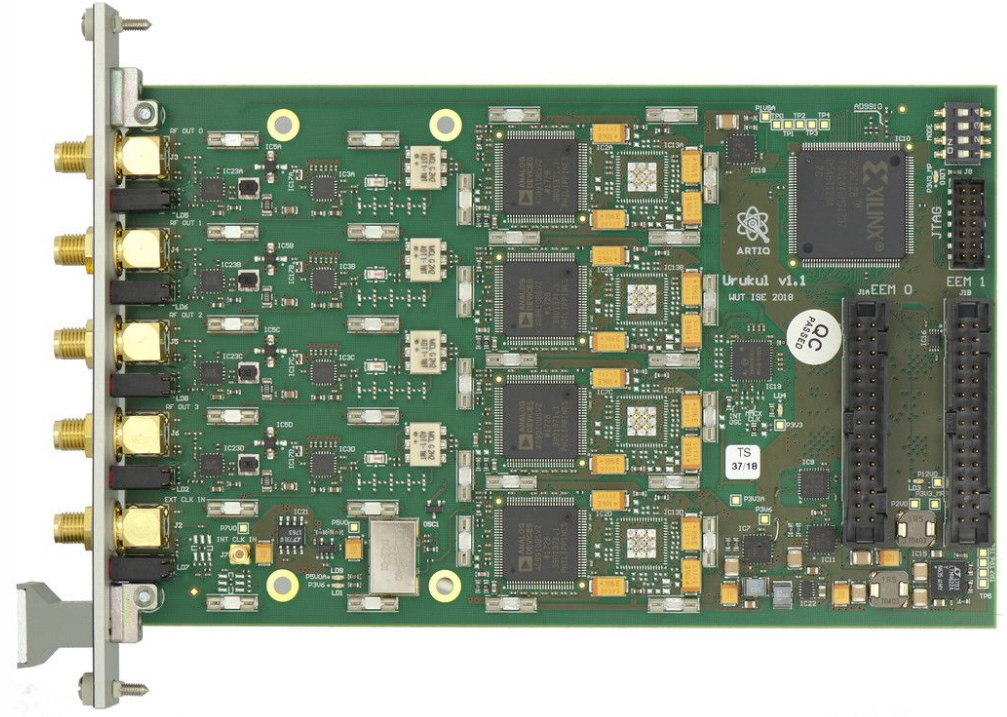
# Use cases

Sampler ADC



+

Urukul DDS



# Use cases

Frame Grabber

+

Camera



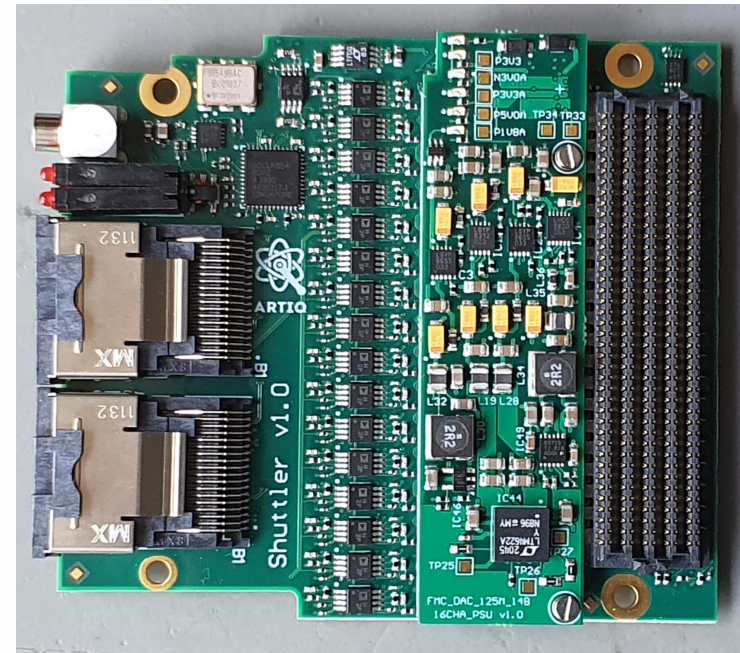
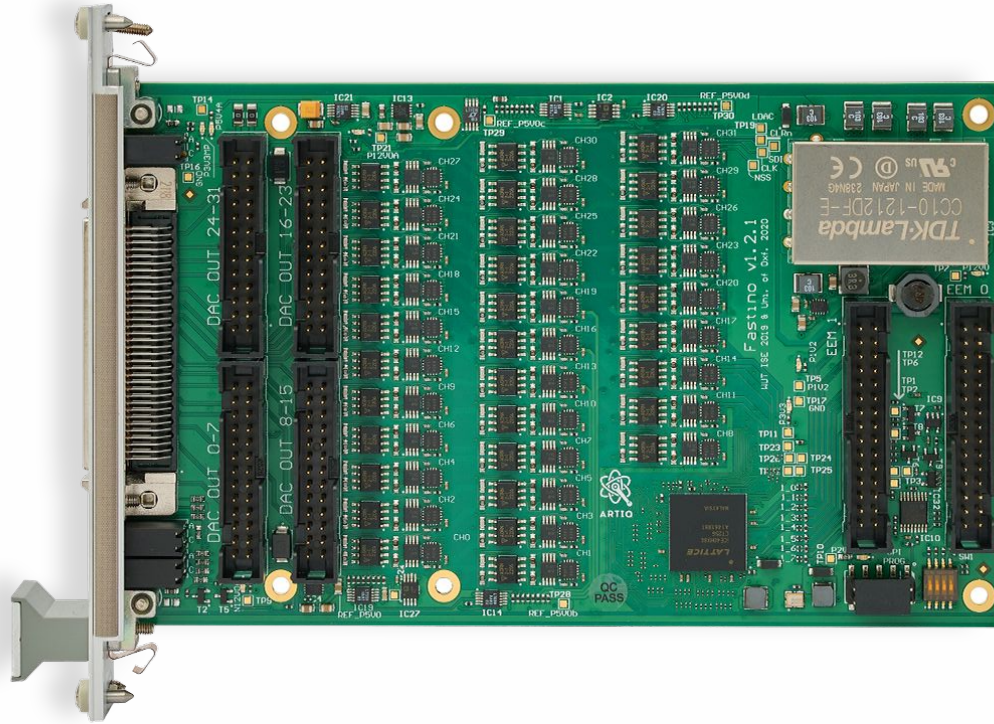


# Use cases

Fastino

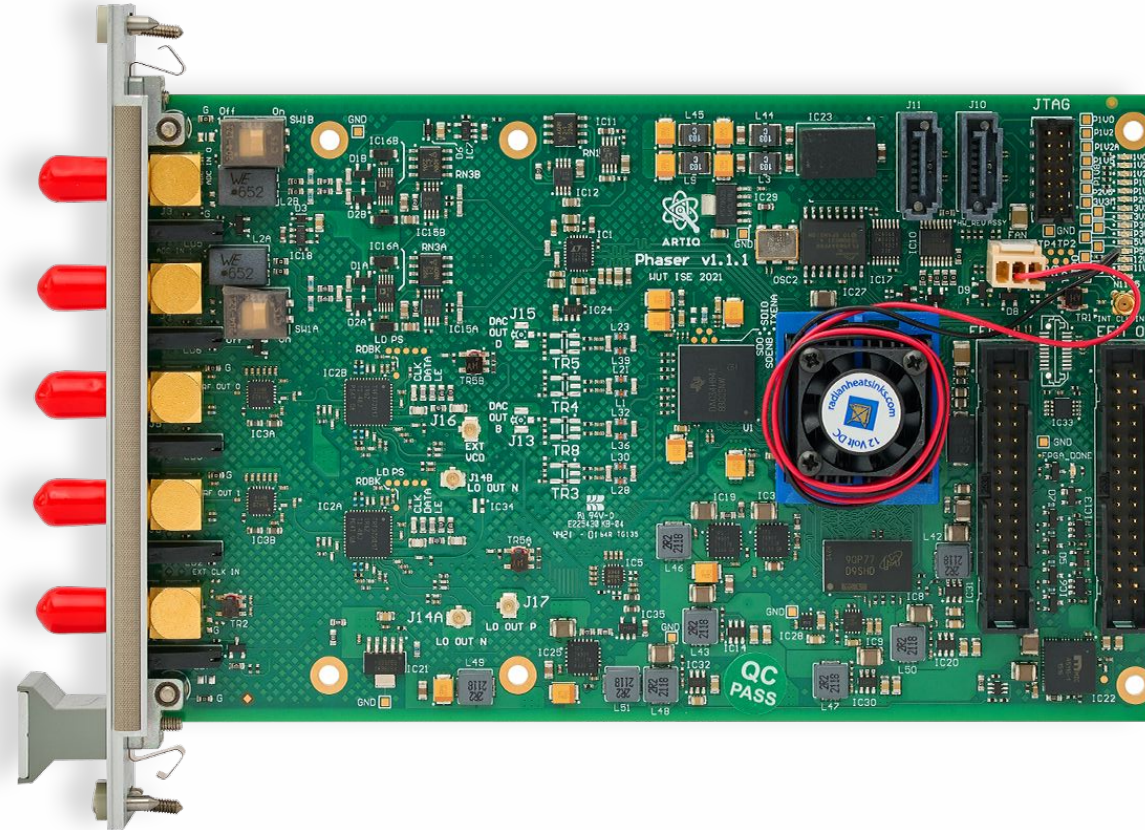
or

Shuttler



# Use cases

## Phaser





# Warsaw University of Technology

[https://github.com/elhep/artiq\\_dax\\_tutorial\\_materials](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](mailto:pawel.kulik@pw.edu.pl)
- [mikolaj.sowinski@pw.edu.pl](mailto:mikolaj.sowinski@pw.edu.pl)

