Warsaw University of Technology

Experiment Control with the ARTIQ/DAX Control Ecosystem

Paweł Kulik
Mikołaj Sowiński
Tomasz Przywózki

IEEE International Conference on Quantum Computing and Engineering - QCE24 Montreal

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

ARTIQ7 manual (used for this tutorial):

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

Organization

Connect to virtual desktop using your personal link

Organization

1. Connect to virtual desktop using your personal link

Glossary

- ARTIQ - Advanced Real-Time Infrastructure for Quantum physics

ARTIQ

Glossary

- ARTIQ
- Sinara - Hardware designed for ARTIQ

Glossary

- ARTIQ
- Sinara
- DAX - Duke ARTIQ Extensions

Glossary

- ARTIQ
- Sinara
- DAX
- (D)RTIO – (Distributed) Real Time Input/Output

Glossary

- ARTIQ
- Sinara
- DAX
- (D)RTIO
- Controller

Glossary

- ARTIQ
- Sinara
- DAX
- (D)RTIO
- Controller
- Satellite - secondary controller

Glossary

- ARTIQ
- Sinara
- DAX
- (D)RTIO
- Controller
- Satellite
- Gateware - firmware for the FPGA

Glossary

- ARTIQ
- Sinara
- DAX
- (D)RTIO
- Controller
- Satellite
- Gateware
- hard real-time - all or nothing

ARTIQ system architecture

What does ARTIQ code look like?

- Python
- Build stage
- Run stage
- Analyze stage

```python

class SystemExample(EnvExperiment):
    def build(self):
        self.setattr_device("core")
        self.setattr_device("core_dma")
        self.fastino = [self.get_device("fastino" + str(i)) for i in range(3)]
        n = 1 << 11
        self.signal = [[0] * 16 for _ in range(n)]
        self.signal2 = [[0] * 16 for _ in range(n)]
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([4*np.sin(200*np.pi*i/n)]*32, self.signal[i])
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([19.9*(i % 2)-9.95]*32, self.signal2[i])

    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()

    @kernel
    def record_waveform(self):
        with self.core_dma.record("fastino_direct"):
            for i in range(len(self.signal)):
                t = now_mu()
                with parallel:
                    self.fastino[0].set_group_mu(0, self.signal[i])
                    self.fastino[1].set_group_mu(0, self.signal[i])
                    self.fastino[2].set_group_mu(0, self.signal[i])
                at_mu(t+self.core.seconds_to_mu(32*8*ns))
                with parallel:
                    self.fastino[0].update(0xffffffff)
                    self.fastino[1].update(0xffffffff)
                    self.fastino[2].update(0xffffffff)

                # Frame period is 392 ns - we have to align to it to have predictable updates
                at_mu(t+self.core.seconds_to_mu(392*1*ns))

        self.core.break_realtime()

    @kernel
    def play_waveform(self):
        dma_handle = self.core_dma.get_handle("fastino_direct")
        self.core.break_realtime()
        for _ in range(10):
            self.core_dma.playback_handle(dma_handle)

    @kernel
    def fastino_output(self, voltage):
        # Set all Fastino channels to some voltage, starting from input parameter and incrementing by 0.2 V
        for i in range(32):
            self.fastino[0].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)
            self.fastino[1].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)
            self.fastino[2].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)

        self.fastino[0].update(0xffffffff)
        self.fastino[1].update(0xffffffff)

    def analyze(self):
        pass
```

Warsaw University of Technology

What does ARTIQ code look like?

- Python
- Build stage
  - Declare devices
  - Add arguments
  - Precalculate variables

```python
class SystemExample(EnvExperiment):
    def build(self):
        self.setattr_device("core")
        self.setattr_device("core_dma")
        self.fastino = [self.get_device("fastino" + str(i)) for i in range(3)]
        n = 1 << 11
        self.signal = [[0] * 16 for _ in range(n)]
        self.signal2 = [[0] * 16 for _ in range(n)]
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([4*np.sin(200*np.pi*i/n)]*32, self.signal[i])
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([19.9*(i % 2)-9.95]*32, self.signal2[i])

    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
```

```python
class SystemExample(EnvExperiment):
    def build(self):
        self.setattr_device("core")
        self.setattr_device("core_dma")
        self.fastino = [self.get_device("fastino" + str(i)) for i in range(3)]
        n = 1 << 11
        self.signal = [[0] * 16 for _ in range(n)]
        self.signal2 = [[0] * 16 for _ in range(n)]
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([4*np.sin(200*np.pi*i/n)]*32, self.signal[i])
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([19.9*(i % 2)-9.95]*32, self.signal2[i])
```

```python
        self.core.break_realtime()

    @kernel
    def play_waveform(self):
        dma_handle = self.core_dma.get_handle("fastino_direct")
        self.core.break_realtime()
        for _ in range(10):
            self.core_dma.playback_handle(dma_handle)

    @kernel
    def fastino_output(self, voltage):
        # Set all Fastino channels to some voltage, starting from input parameter and incrementing by 0.2 V
        for i in range(32):
            self.fastino[0].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)
            self.fastino[1].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)
            self.fastino[2].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)

        self.fastino[0].update(0xffffffff)
        self.fastino[1].update(0xffffffff)

    def analyze(self):
        pass
```

Warsaw University of Technology

What does ARTIQ code look like?

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

```python
    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()
```

```python
class SystemExample(EnvExperiment):
    def build(self):
        self.setattr_device("core")
        self.setattr_device("core_dma")
        self.fastino = [self.get_device("fastino" + str(i)) for i in range(3)]
        n = 1 << 11
        self.signal = [[0] * 16 for _ in range(n)]
        self.signal2 = [[0] * 16 for _ in range(n)]
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([4*np.sin(200*np.pi*i/n)]*32, self.signal[i])
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([19.9*(i % 2)-9.95]*32, self.signal2[i])

    @kernel
    def run(self):
        self.core.break_realtime()
```

```python
    @kernel
    def fastino_output(self, voltage):
        # Set all Fastino channels to some voltage, starting from input parameter and incrementing by 0.2 V
        for i in range(32):
            self.fastino[0].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)
            self.fastino[1].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)
            self.fastino[2].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)

        self.fastino[0].update(0xffffffff)
        self.fastino[1].update(0xffffffff)

    def analyze(self):
        pass
```

Warsaw University of Technology

What happens with my code?

```python
    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()
```

- 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 does ARTIQ code look like?

- Python
- Build stage
- Run stage
- Analyze stage
  - Data analysis
  - Saving results to the database

```python
    def analyze(self):
        pass
```

```python

class SystemExample(EnvExperiment):
    def build(self):
        self.setattr_device("core")
        self.setattr_device("core_dma")
        self.fastino = [self.get_device("fastino" + str(i)) for i in range(3)]
        n = 1 << 11
        self.signal = [[0] * 16 for _ in range(n)]
        self.signal2 = [[0] * 16 for _ in range(n)]
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([4*np.sin(200*np.pi*i/n)]*32, self.signal[i])
        for i in range(n):
            self.fastino[0].voltage_group_to_mu([19.9*(i % 2)-9.95]*32, self.signal2[i])

    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()

    @kernel
    def record_waveform(self):
        with self.core_dma.record("fastino_direct"):
            for i in range(len(self.signal)):
                t = now_mu()
                with parallel:
                    self.fastino[0].set_group_mu(0, self.signal[i])
                    self.fastino[1].set_group_mu(0, self.signal[i])
                    self.fastino[2].set_group_mu(0, self.signal[i])
                at_mu(t+self.core.seconds_to_mu(32*8*ns))
                with parallel:
                    self.fastino[0].update(0xffffffff)
                    self.fastino[1].update(0xffffffff)
                    self.fastino[2].update(0xffffffff)

                # Frame period is 392 ns - we have to align to it to have predictable updates
                at_mu(t+self.core.seconds_to_mu(392*1*ns))

        self.core.break_realtime()

    @kernel
    def play_waveform(self):
        dma_handle = self.core_dma.get_handle("fastino_direct")
        self.core.break_realtime()
        for _ in range(10):
            self.core_dma.playback_handle(dma_handle)

    @kernel
    def fastino_output(self, voltage):
        # Set all Fastino channels to some voltage, starting from input parameter and incrementing by 0.2 V
        for i in range(32):
            self.fastino[0].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)
            self.fastino[1].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)
            self.fastino[2].set_dac(i, voltage + 0.2 * i)
            delay(100 * us)

        self.fastino[0].update(0xffffffff)
        self.fastino[1].update(0xffffffff)

    def analyze(self):
        pass
```

Warsaw University of Technology

What happens with my code?

```python
    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()
```

- 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?

```python
    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()
```

- 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?

```python
    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()
```

- 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?

```python
    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()
```

- 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?

```python
    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()
```

- 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?

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

```python
    @kernel
    def run(self):
        self.core.break_realtime()
        self.core.reset()
        self.init()
        self.core.break_realtime()

        # No interpolation
        self.record_waveform()
        self.play_waveform()

    @kernel
    def init(self):
        for fastino in self.fastino:
            fastino.init()
            fastino.set_hold(0xffffffff)
            delay(8*ns)
            fastino.set_group(0, [0.*V]*32)
            delay(1*us)
            fastino.update(0xffffffff)
            delay(10*ms)
        self.core.break_realtime()
```

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

Devices that will be used

- Kasli
- DIO TTL
- Urukul DDS
- Phaser AWG
  - 2 channel 1.25 GS/s RF generator
  - baseband and upconverter variants
  - https://github.com/sinara-hw/Phaser/wiki

Warsaw University of Technology

What can I do in the FPGA?

- Subset of Python

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

Warsaw University of Technology

What can I do in the FPGA?

- Subset of Python
- Conditionals (if)

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

Warsaw University of Technology

What can I do in the FPGA?

- Subset of Python
- Conditionals (if)
- Loops, iterating over lists

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

Warsaw University of Technology

What can I do in the FPGA?

- Subset of Python
- Conditionals (if)
- Loops, iterating over lists
- Exceptions

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

Warsaw University of Technology

What can I do in the FPGA?

- Subset of Python
- Conditionals (if)
- Loops, iterating over lists
- Exceptions
- Timing

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

Warsaw University of Technology

Organization

Organization

artiq_tutorial

system_<a..f>

repository

user<00..11>

artiq

Warsaw University of Technology

Organization

Organization

Organization

Applications ARTIQ Dashboard - System A Thu 13:14

ARTIQ Dashboard - System A

Applet: User 3 Scope View

repo:user03/artiq/Initialize

Recompute all arguments | Load HDF5

Pipeline: main

Submit

Explorer

Revision: N/A

- user00
- user01
- user02
- user03
  - artiq
    - DMAExcercise
    - Initialize
    - ParallelExcercise
    - Timing1Excercise
    - Timing2Excercise
  - 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

Schedule | Log

Organization

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

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

Priority: 0 Flush

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

Submit

Explorer

Revision: N/A

- user01
- user02
- user03
  - artiq
    - DMAExcercise
    - Initialize
    - ParallelExcercise
    - Timing1Excercise
    - Timing2Excercise
  - 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

Timing examples

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

Warsaw University of Technology

Timing examples

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

Warsaw University of Technology

Timing examples

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

Warsaw University of Technology

Timing examples

```
for i in range(len(self.signal)):
    t = now_mu()
    with parallel:
        self.fastino[0].set_group_mu(0, self.signal[i])
        self.fastino[1].set_group_mu(0, self.signal[i])
        self.fastino[2].set_group_mu(0, self.signal[i])
    at_mu(t+self.core.seconds_to_mu(32*8*ns))
    with parallel:
        self.fastino[0].update(0xffffffff)
        self.fastino[1].update(0xffffffff)
        self.fastino[2].update(0xffffffff)

    # Frame period is 392 ns - we have to align to it to have predictable updates
    at_mu(t+self.core.seconds_to_mu(392*1*ns))
```

Timing examples - exercise Timing1

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

Goal: create 2 pulses using both TTL methods (on/off and pulse) and delay function.

TTL methods:

- ttl.on()
- tll.off()
- ttl.pulse(*duration*) - **hidden delay inside!**

Timing functions:

- delay(*duration*)

Dashboard arguments:

- FirstPulseWidth
- DelayToNextPulse
- SecondPulseWidth

min. values: 10 ns
max. values: 400 ns

Timing examples - exercise Timing2

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

```
t = now_mu()
at_mu(t + self.core.seconds_to_mu(2 * us))
```

Goal: create 2 pulses using both TTL methods (on/off and pulse), now_mu and at_mu functions

TTL methods:

- ttl.on()
- tll.off()
- ttl.pulse(*duration*) - **hidden delay inside!**

Timing functions:

- now_mu() - returns current time pointer in machine units
- at_mu(time_in_mu) - jump to exact time in machine units
- self.core.seconds_to_mu(time_value) - convert time value in second to machine units

Dashboard arguments:

- FirstPulseWidth
- DelayToNextPulse
- SecondPulseWidth

min. values: 10 ns
max. values: 400 ns

Warsaw University of Technology

RTIO Underflow + Exceptions

```
try:
    ttl.on()
except RTIOUnderflow:
    # try again at the next mains cycle
    delay(16.6667*ms)
    ttl.on()
```

Warsaw University of Technology

RTIO Underflow + Exceptions

Warsaw University of Technology

RTIO Underflow + Exceptions - exercise Timing3

TTL methods:

- ttl.on()
- ttl.off()
- ttl.pulse(*duration*)

Timing functions:

- delay(*duration*)

Dashboard arguments:

- Delay - time between next edge event

min. value: 10 ns
max. value: 1000 ns
working min. value: ~390ns

Goal: create 10k iterations for loop. In each loop switch TTL on and off with delay between events.

Warsaw University of Technology

What can I do in the FPGA?

- Subset of Python
- Conditionals (if)
- Loops, iterating over lists
- Exceptions
- Timing
- DMA - prerecording sequences of events

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

Warsaw University of Technology

DMA

```
@kernel
def record(self):
    with self.core_dma.record("pulses"):
        # all RTIO operations now go to the "pulses"
        # DMA buffer, instead of being executed immediately.
        for i in range(50):
            self.ttl0.pulse(100*ns)
            delay(100*ns)

@kernel
def run(self):
    self.core.reset()
    self.record()
    # prefetch the address of the DMA buffer
    # for faster playback trigger
    pulses_handle = self.core_dma.get_handle("pulses")
    self.core.break_realtime()
    while True:
        # execute RTIO operations in the DMA buffer
        # each playback advances the timeline by 50*(100+100) ns
        self.core_dma.playback_handle(pulses_handle)
```

TTL methods:

- ttl.on()
- tll.off()
- ttl.pulse(*duration*)

Timing functions:

- delay(*duration*)

Duration examples:
4 * us; 400 * ns

Dashboard arguments:

- Delay - time between next edge event

min. value: 10 ns
max. value: 1000 ns
working min. value: ~195ns

Goal: create the same loop as in timing2 exercise but using DMA (100 iterations of loop + 100 iterations over DMA record)

What can I do in the FPGA?

- Subset of Python
- Conditionals (if)
- Loops, iterating over lists
- Exceptions
- Timing
- DMA
- RPC

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

Warsaw University of Technology

RPC - remote procedure call

```
def input_led_state() -> TBool:
    return input("Enter desired LED state: ") == "1"

class LED(EnvExperiment):
    def build(self):
        self.setattr_device("core")
        self.setattr_device("led")

    @kernel
    def run(self):
        self.core.reset()
        s = input_led_state()
        self.core.break_realtime()
        if s:
            self.led.on()
        else:
            self.led.off()
```

What can I do in the FPGA?

- Subset of Python
- Conditionals (if)
- Loops, iterating over lists
- Exceptions
- Timing
- DMA - prerecording sequences of events
- RPC
- Parallel operation

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

Warsaw University of Technology

Parallel and sequential blocks

```
for i in range(1000000):
    with parallel:
        self.ttl0.pulse(2*us)
        self.ttl1.pulse(4*us)
    delay(4*us)
```

Warsaw University of Technology

Parallel and sequential blocks

```
for i in range(1000000):
    with parallel:
        self.ttl0.pulse(2*us)
        self.ttl1.pulse(4*us)
    delay(4*us)
```

```
for i in range(1000000):
    with parallel:
        with sequential:
            self.ttl0.pulse(2*us)
            delay(1*us)
            self.ttl0.pulse(1*us)
        self.ttl1.pulse(4*us)
    delay(4*us)
```

Exercise: Synchronised TTL and DDS pulses

TTL methods:

- **ttl.pulse(*duration*)**

Timing functions:

- **delay(*duration*)**

Duration examples:
4 * us; 400 * ns

Urukul channels methods:

- channel.init()
- channel.set(
  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 attenuator in SI units [0, 31.5 dB]
- **channel.sw.on()**
- **channel.sw.off()**

```
for i in range(1000000):
    with parallel:
        with sequential:
            self.ttl0.pulse(2*us)
            delay(1*us)
            self.ttl0.pulse(1*us)
        self.ttl1.pulse(4*us)
    delay(4*us)
```

Goal: At the same time start ttl.pulse(200*ns) and switch on both Urukul channels. Switch off first Urukul channel after 200 ns and second after 400 ns. Run second TTL pulse (100 ns wide) exactly 100 ns after first pulse. Manipulate time only with delay function.

Use parallel and sequential blocks.

Warsaw University of Technology

Exercise: Synchronised TTLs and DDS pulses

Warsaw University of Technology

Demonstration: Phaser AWG

Warsaw University
of Technology

ROHDE&SCHWARZ
RTE2304 · Digital Oscilloscope · 2.5 GSa/s
Auto
Run
Sample
Help
Vertical
Logic
Trigger
Acquire
History
5 V/
500 mV/
500 mV/
200 mV/
20 dBm/

Demonstration: Phaser AWG

```python
def build(self):
    self.setattr_device("core")
    self.setattr_device("core_dma")
    self.setattr_device("phaser0")

    self.center_f = 97 * MHz

    freq_slopes = [
        np.linspace(2.8 * MHz, 1 * MHz, 100),
        np.linspace(2.9 * MHz, 2 * MHz, 100),
        np.full(100, 3 * MHz),
        np.linspace(3.1 * MHz, 4 * MHz, 100),
        np.linspace(3.2 * MHz, 5 * MHz, 100),
    ]

    amp_slopes = [
        np.linspace(0.375, 0.01, 100) * 0.9,
        np.linspace(0.10, 0.25, 100) * 0.9,
        np.linspace(0.05, 0.48, 100) * 0.9,
        np.linspace(0.10, 0.25, 100) * 0.9,
        np.linspace(0.375, 0.01, 100) * 0.9,
    ]

    # Make sure that summed amps never exceed 1.0 (full scale)
    assert all(sum(amps) <= 1.0 for amps in zip(*amp_slopes))

    self.ftw = [np.concatenate([slope, slope[::-1]]) for slope in freq_slopes]
    self.amps = [np.concatenate([slope, slope[::-1]]) for slope in amp_slopes]
    self.length = len(self.ftw[0])
```

Demonstration: Phaser AWG

```
@kernel
def init(self):
    self.core.reset()
    self.core.break_realtime()

    self.phaser0.init()
```

```
@kernel
def run(self):
    self.init()

    phaser = self.phaser0

    phaser.channel[0].set_att(0 * dB)
    phaser.channel[0].set_duc_frequency(self.center_f)
    phaser.channel[0].set_duc_phase(0.25)
    phaser.channel[0].set_duc_cfg()

    delay(0.1 * ms)
    phaser.duc_stb()
    delay(0.1 * ms)

    while True:
        for i in range(self.length):
            for osc in range(5):
                phaser.channel[0].oscillator[osc].set_frequency(self.ftw[osc][i])
                phaser.channel[0].oscillator[osc].set_amplitude_phase(
                    self.amps[osc][i], phase=0.25)
                delay(5 * ms)
```

Demonstration: Phaser AWG

```
@kernel
def init(self):
    self.core.reset()
    self.core.break_realtime()

    self.phaser0.init()

@kernel
def run(self):
    self.init()

    phaser = self.phaser0

    phaser.channel[0].set_att(0 * dB)
    phaser.channel[0].set_duc_frequency(self.center_f)
    phaser.channel[0].set_duc_phase(0.25)
    phaser.channel[0].set_duc_cfg()

    delay(0.1 * ms)
    phaser.duc_stb()
    delay(0.1 * ms)

    while True:
        for i in range(self.length):
            for osc in range(5):
                phaser.channel[0].oscillator[osc].set_frequency(self.ftw[osc][i])
                phaser.channel[0].oscillator[osc].set_amplitude_phase(
                    self.amps[osc][i], phase=0.25)
                delay(5 * ms)
```

Demonstration: Phaser AWG

```
@kernel
def init(self):
    self.core.reset()
    self.core.break_realtime()

    self.phaser0.init()

@kernel
def run(self):
    self.init()

    phaser = self.phaser0

    phaser.channel[0].set_att(0 * dB)
    phaser.channel[0].set_duc_frequency(self.center_f)
    phaser.channel[0].set_duc_phase(0.25)
    phaser.channel[0].set_duc_cfg()

    delay(0.1 * ms)
    phaser.duc_stb()
    delay(0.1 * ms)

    while True:
        for i in range(self.length):
            for osc in range(5):
                phaser.channel[0].oscillator[osc].set_frequency(self.ftw[osc][i])
                phaser.channel[0].oscillator[osc].set_amplitude_phase(
                    self.amps[osc][i], phase=0.25)
                delay(5 * ms)
```

Demonstration: Phaser AWG

```
@kernel
def init(self):
    self.core.reset()
    self.core.break_realtime()

    self.phaser0.init()

@kernel
def run(self):
    self.init()

    phaser = self.phaser0

    phaser.channel[0].set_att(0 * dB)
    phaser.channel[0].set_duc_frequency(self.center_f)
    phaser.channel[0].set_duc_phase(0.25)
    phaser.channel[0].set_duc_cfg()

    delay(0.1 * ms)
    phaser.duc_stb()
    delay(0.1 * ms)

    while True:
        for i in range(self.length):
            for osc in range(5):
                phaser.channel[0].oscillator[osc].set_frequency(self.ftw[osc][i])
                phaser.channel[0].oscillator[osc].set_amplitude_phase(
                    self.amps[osc][i], phase=0.25)
                delay(5 * ms)
```

Demonstration: Phaser AWG

```
@kernel
def init(self):
    self.core.reset()
    self.core.break_realtime()

    self.phaser0.init()

@kernel
def run(self):
    self.init()

    phaser = self.phaser0

    phaser.channel[0].set_att(0 * dB)
    phaser.channel[0].set_duc_frequency(self.center_f)
    phaser.channel[0].set_duc_phase(0.25)
    phaser.channel[0].set_duc_cfg()

    delay(0.1 * ms)
    phaser.duc_stb()
    delay(0.1 * ms)

    while True:
        for i in range(self.length):
            for osc in range(5):
                phaser.channel[0].oscillator[osc].set_frequency(self.ftw[osc][i])
                phaser.channel[0].oscillator[osc].set_amplitude_phase(
                    self.amps[osc][i], phase=0.25)
                delay(5 * ms)
```

Use cases

Sampler ADC + Urukul DDS

Warsaw University of Technology

Use cases

Frame Grabber + Camera

Use cases

Fastino or Shuttler

Warsaw University of Technology

Warsaw University of Technology

https://github.com/elhep/artiq_dax_tutorial_materials