Timing control in ARTIQ

Sébastien Bourdeauducq

M-Labs Ltd – https://m-labs.hk

Background

- ARTIQ “kernels” are Python programs compiled and executed on the core device.
- CPU with tightly coupled I/O timing gateware (“RTIO core”).
- High resolution (nanosecond).
- Low latency (microsecond).

ARTIQ

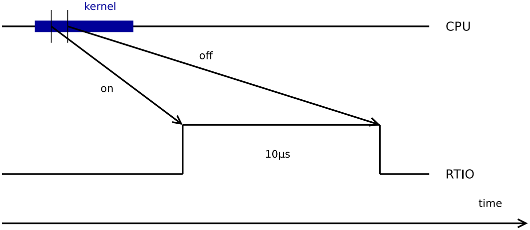
The basics

- The CPU maintains a time cursor used as timestamp to program all RTIO commands.
- The time cursor can be advanced using `delay()` and `delay_mu()`.
- The absolute position of the time cursor can be retrieved using `now_mu()` and set using `at_mu()`.
- Gateware looks at the timestamps of programmed RTIO commands, and executes them at the appropriate time.
- This guarantees "all or nothing" excellent timing precision.
- Absolute RTIO timestamps are referenced to the core device startup (gateware time counter keeps running across experiments).

The basics

A precisely timed 10μs pulse.

```
ttl.on()
delay(10*us)
ttl.off()
```

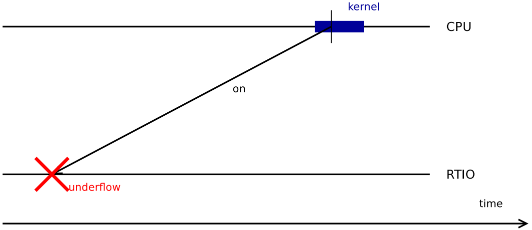


Why the *_mu functions?

- Absolute timestamps can be large numbers. They are represented internally as 64-bit integers.
- Conversions between such a large number and floating point in seconds can cause loss of precision.
- When computing the difference of absolute timestamps, use `mu_to_seconds(t2-t1)`, not `mu_to_seconds(t2)-mu_to_seconds(t1)`.

Underflows

An RTIO command must be programmed with a timestamp in the future.

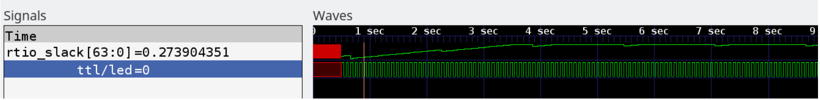


The core language supports exceptions

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

Tracking down underflows

- Exception backtraces tell you where underflows have occured.
- Analyzer supports plotting of RTIO slack (at submission of RTIO command, difference between time cursor and physical time).



Pulse method

```
ttl.on()
delay(10*us)
ttl.off()
```

is equivalent to:

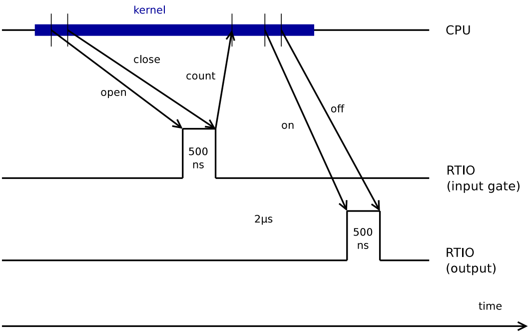
```
ttl.pulse(10*us)
```

The `pulse` method advances the time cursor. Other methods such as `on`, `off`, and the `set` method of DDSes do not. The latter are called “zero-duration” methods.

Input

Count the rising edges occuring during a precisely timed 500ns interval. Output pulse if more than 20 were received.

```
input.gate_rising(500*ns)
if input.count() > 20:
    delay(2*us)
    output.pulse(500*ns)
```



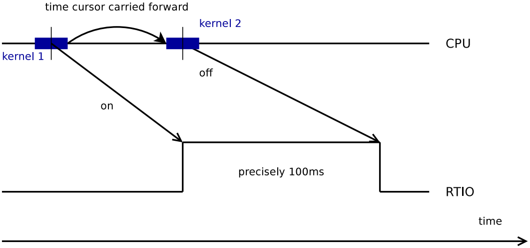
Overflow error

- The gateware buffers input events received while the input gate is open.
- It keeps them in a FIFO until the CPU reads them out via `count` (or `timestamp_mu`).
- If the FIFO is full and another event is coming, it causes an overflow error.
- The `RTIOOverflow` exception is raised by the readout method.

Seamless handover

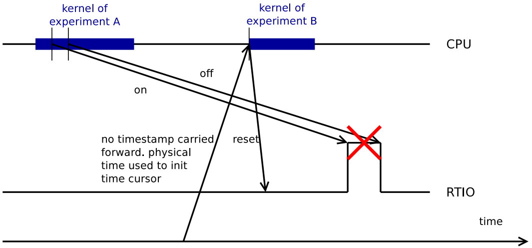
```
@kernel                 @kernel
def kernel1():          def kernel2():
  ttl.on()                ttl.off()
  delay(100*ms)

def run():
  kernel1(); kernel2()
```



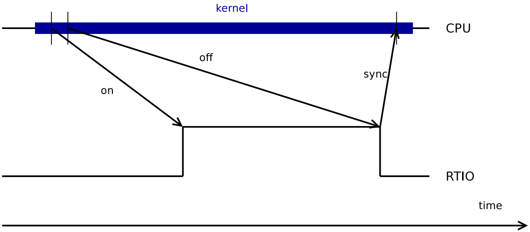
Resets

- Problem: previous kernel sets time cursor far in the future, locks system.
- Solution: when switching experiments, clear RTIO FIFOs and reinit time cursor.



Synchronization operation

- Problem: kernel returns before its last RTIO command is executed, next experiment cancels it.
- Solution: sync command.



Issue 425: an alternate approach (2.0)?

- Maintain seamless handover between experiments.
- Trust that experiments end with the time cursor in a reasonable position.
- Sanity checks on the time cursor after an experiment ends.
- Resets triggered by user or failed sanity checks to recover from RTIO breakage.
- Case of the idle kernel – only run when no experiment present plus time cursor position well below physical time.