Zephyr® Project
Developer Summit 2022

# RTIO: A BLOCK STREAM API

Or io_uring for Zephyr

Tom Burdick, Intel Corporation

# Problems? Always!

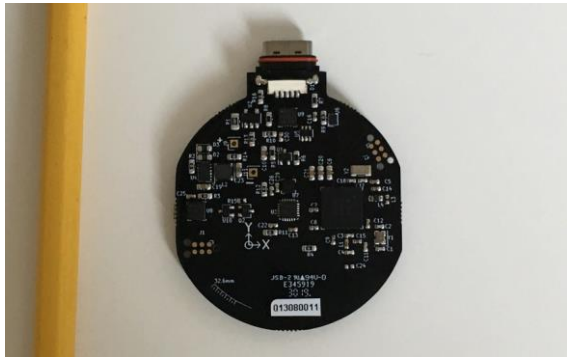
- Blocking I/O requires a thread
- Async calls are underdefined
- No application control of DMA
- Callbacks are great, until you chain them
- Copying is wasteful

# The Boss: The problem Project

# The Boss: The problem project

- Baseball pitch training aid with an nRF52840
- Many sensors on SPI buses
- From arm cocked to release, 0.03s, total pitch ~2-3s
- RAM for sensor data
- CPU time for pattern matching
- Timing is everything, no shared sensor clock

**Didn't work out of the box**



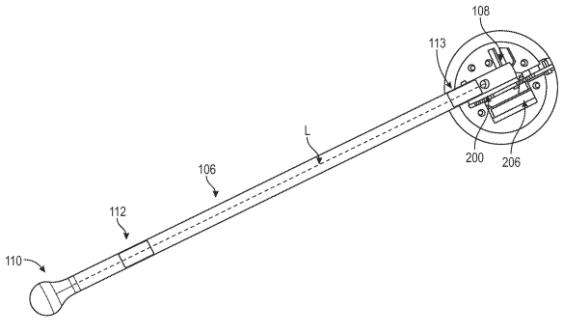



FIG. 2B

# The Boss: The problem project

- Async ISR started transfer saved RAM and Time
- DMA coping sensor FIFOs frees up CPU time
- ISR timestamping improves accuracy

**Required forking Zephyr**

# The Essence of the Problem

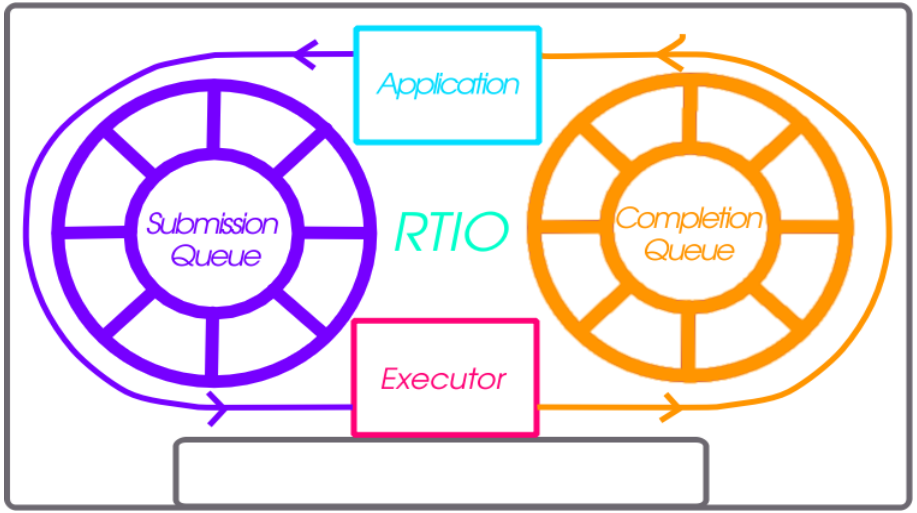
**No way to do I/O, the way I wanted, when I wanted.**

# Solution: Queues, its always Queues

- Peripherals are already concurrent, no need for a software thread.
- Message the hardware, get a message back.
- Submit requests
- Receive completions
- Matches up with DMA

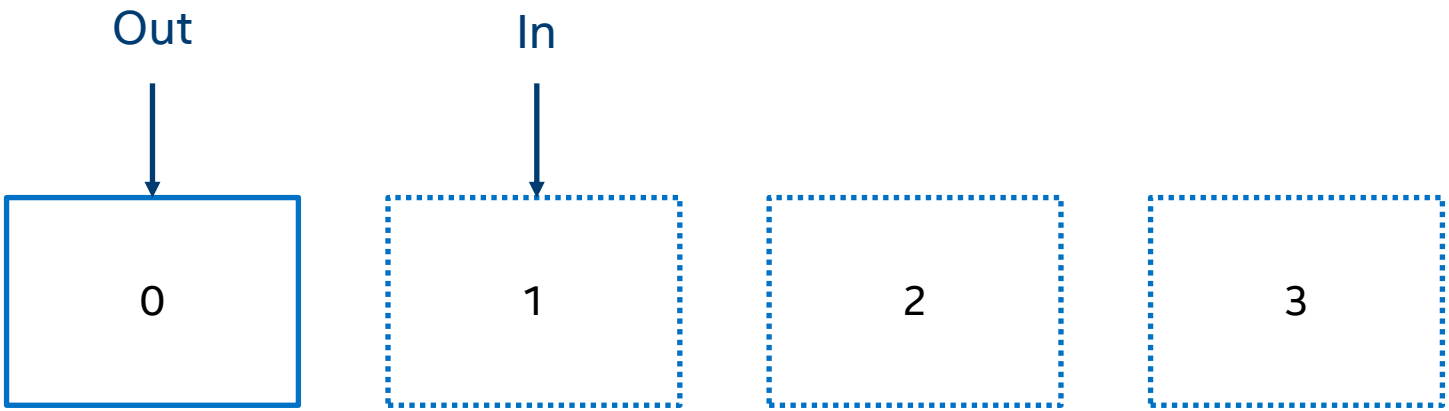
# RTIO: A pair queues

- A context of asynchronous execution across peripherals
- Input is a queue of submissions, output is a queue of completions
- Pluggable executor for scheduling
- Pluggable iodev for handling submissions
- Modeled on Linux's io_uring, reusing the terms and some ideas



# RTIO: Simple Submissions

- Ordering given by queue order
- Failures do not cascade

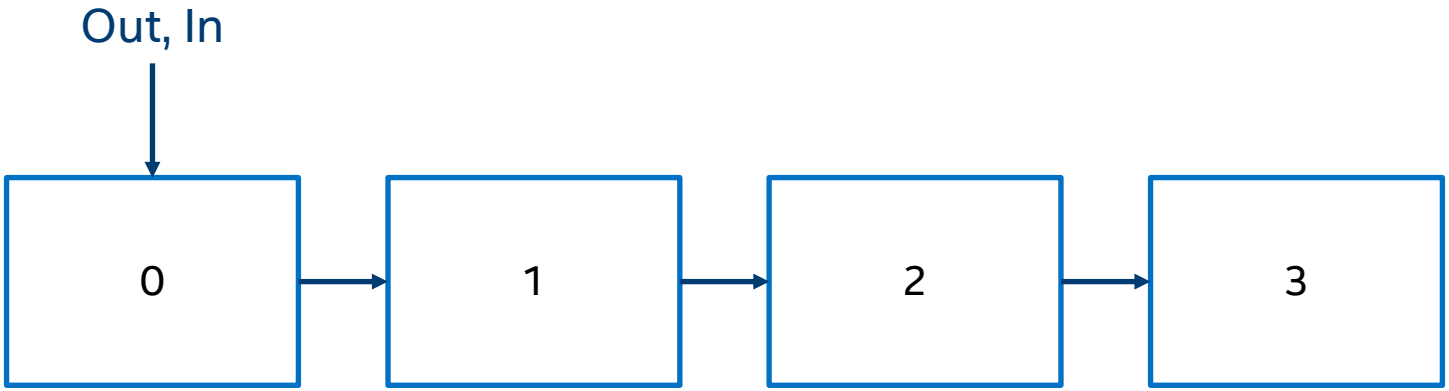


# RTIO: Simple Submission

```c
void test_rtio_simple_(struct rtio *r)
{
    int res;
    uintptr_t userdata[2] = {0, 1};
    struct rtio_sqe *sqe;
    struct rtio_cqe *cqe;

    rtio_iodev_test_init(&iodev_test_simple);

    TC_PRINT("setting up single no-op\n");
    sqe = rtio_spsc_acquire(r->sq);
    zassert_not_null(sqe, "Expected a valid sqe");
    rtio_sqe_prep_nop(sqe, (struct rtio_iodev *)&iodev_test_simple, &userdata[0]);

    TC_PRINT("submit with wait\n");
    res = rtio_submit(r, 1);
    zassert_ok(res, "Should return ok from rtio_execute");

    cqe = rtio_spsc_consume(r->cq);
    zassert_not_null(cqe, "Expected a valid cqe");
    zassert_ok(cqe->result, "Result should be ok");
    zassert_equal_ptr(cqe->userdata, &userdata[0], "Expected userdata back");
    rtio_spsc_release(r->cq);
}
```

# RTIO: Linked Submissions

- Enforced ordering across devices
- Failures cascade

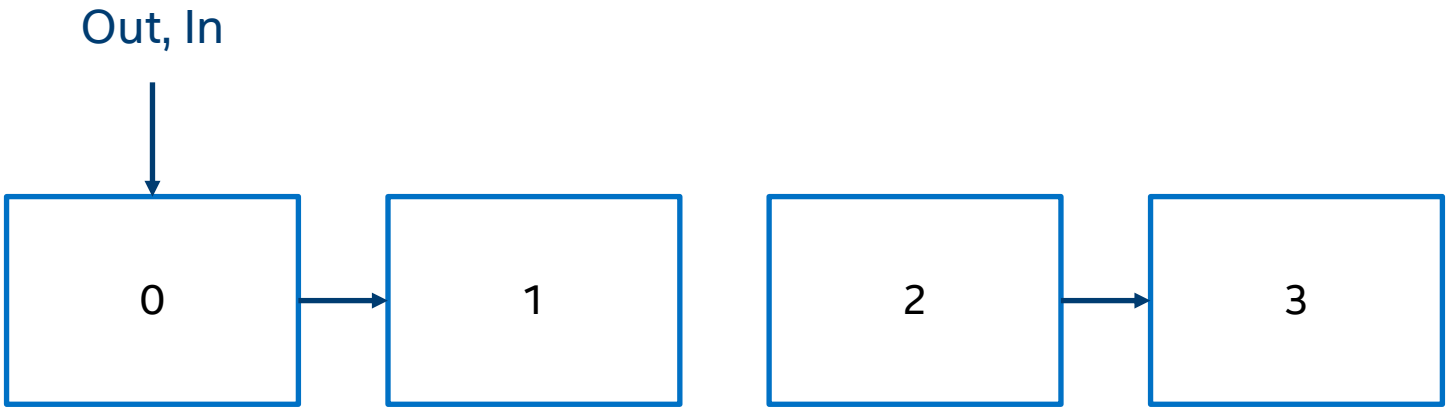


# RTIO: Linked Submissions

```c
void test_rtio_chain_(struct rtio *r)
{
    int res;
    uintptr_t userdata[4] = {0, 1, 2, 3};
    struct rtio_sqe *sqe;
    struct rtio_cqe *cqe;

    for (int i = 0; i < 4; i++) {
        sqe = rtio_spsc_acquire(r->sq);
        zassert_not_null(sqe, "Expected a valid sqe");
        rtio_sqe_prep_nop(sqe, (struct rtio_iodev *)&iodev_test_chain[i % 2],
                &userdata[i]);
        sqe->flags |= RTIO_SQE_CHAINED;
    }

    /* Clear the last one */
    sqe->flags = 0;

    res = rtio_submit(r, 4);
    zassert_ok(res, "Should return ok from rtio_execute");
    zassert_equal(rtio_spsc_consumable(r->cq), 4, "Should have 4 pending completions");

    for (int i = 0; i < 4; i++) {
        TC_PRINT("consume %d\n", i);
        cqe = rtio_spsc_consume(r->cq);
        zassert_not_null(cqe, "Expected a valid cqe");
        zassert_ok(cqe->result, "Result should be ok");
        zassert_equal_ptr(cqe->userdata, &userdata[i], "Expected in order completions");
        rtio_spsc_release(r->cq);
    }
}
```

# RTIO: Multiple Linked Entries in Submission Queue

- Chains may be executed concurrently
- Failures contained to each chain
- SPI0 reading Sensor0, SPI1 reading Sensor1, Concurrent Chains!



# RTIO: Does it solve the Problem?

- Predefined operations enable ISR sequences
- Applications don't need to directly write ISR logic
- Still get the benefits of reduced thread counts and precise timing
- Submissions may include DMA requests for larger transfers
- Executor and iodev could transform reads/writes into DMA operations based on flags or policy

**No Zephyr Fork Required**

# New Solutions come with New Problems

- Current bus device APIs abstract scatter gather already
- Current async function behavior varied
- What happens when a device is busy?

## Start Somewhere Useful

- Prove at a Higher Level (ADC, Sensors, Video, Audio)
- Iterate on Design
- Expand Downward (I2C/SPI/I3C/I2S/CSI)

# Discuss and Find Out More

- GitHub Handle: teburd
- GitHub Pull: https://github.com/zephyrproject-rtos/zephyr/pull/44999
- Discord Channel: https://discordapp.com/channels/720317445772017664/902932500672827432
- Discord Handle: tomb

Zephyr® Project
Developer Summit 2022

# DISCUSSION!