Taking Your Hardware To Production With Zephyr

Using elements of the RTOS and Ecosystem to build better products

golioth

The journey of a device going to production

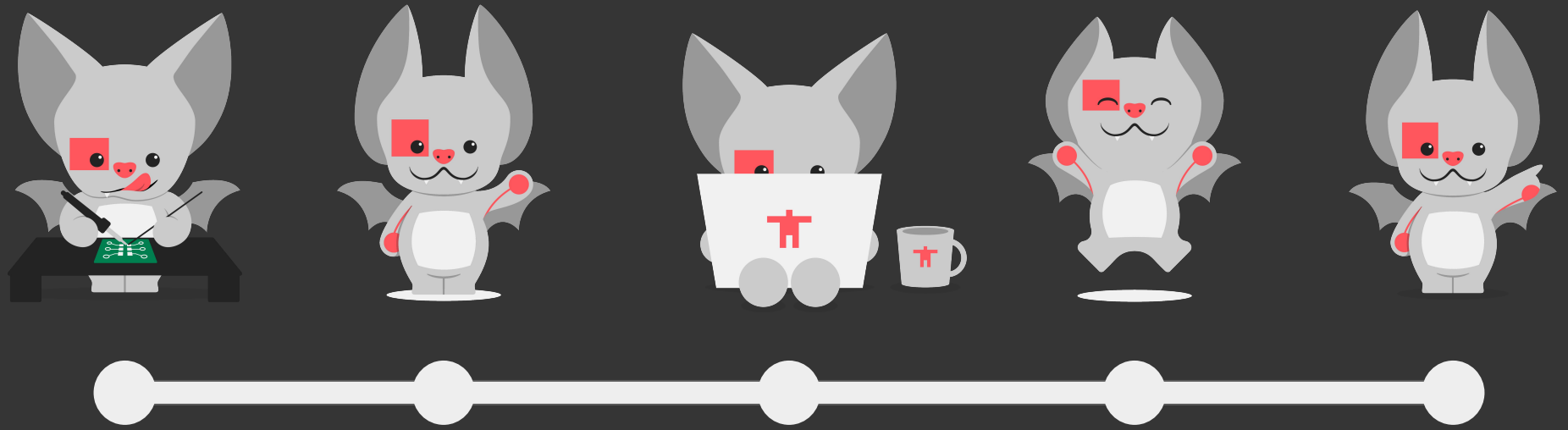
Early Prototyping On Devboards

Custom HW, First Pilot

First Devices in Production

Scaling Production

Maintaining Scaled Fleet



Taking a Zephyr device to production

SAME PROBLEMS, NEW TOOLS

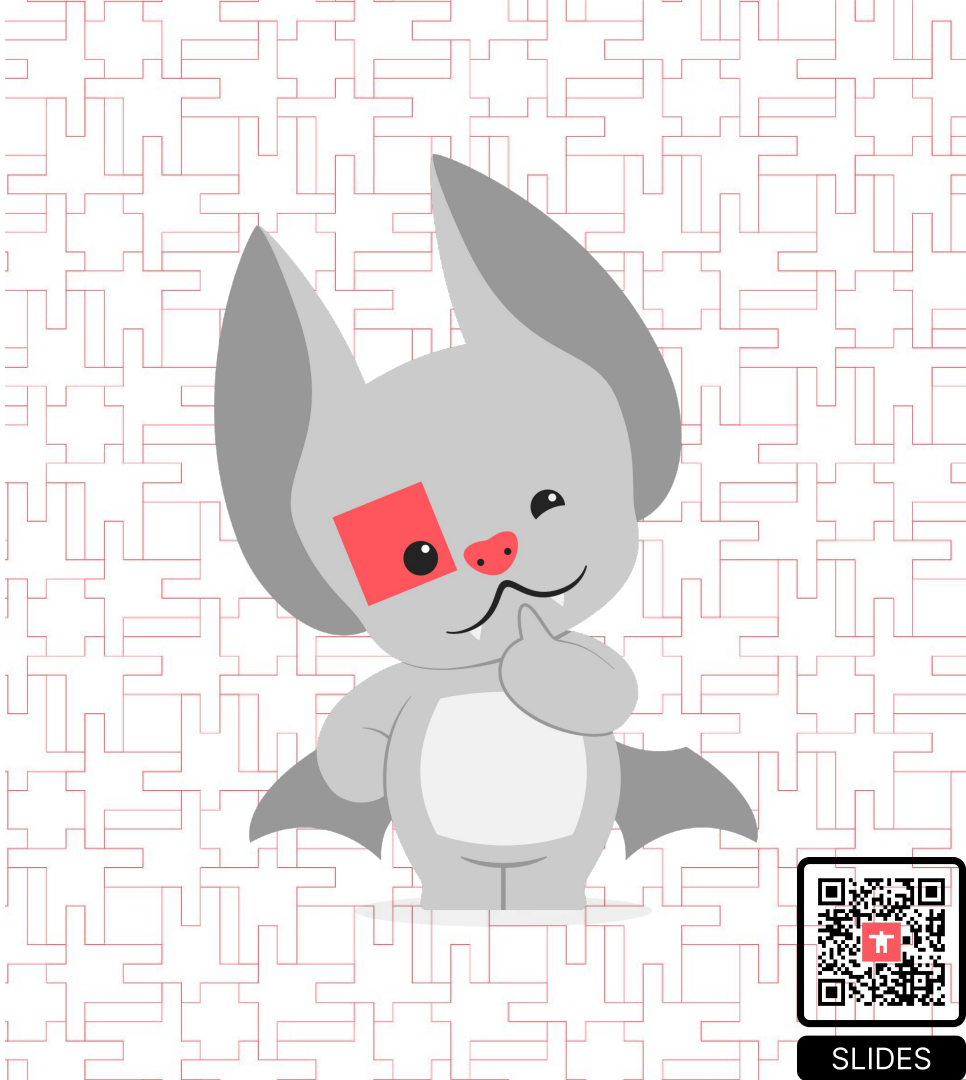
- Building a device using Zephyr RTOS doesn't change the nature of the problem space, but it does give you more tools in your toolbox for solving them, on the bench and in the factory

METHODS MIGHT WORK AT DIFFERENT TIMES FOR YOU

- Every product development cycle is different, so many of the things shown today could be implemented at a different point in the process.
- The most successful productization cycles move testing as far forward as possible

THIS IS BIG AREA OF STUDY

- Zephyr has so many ways of doing things. We will have links out to lots of resources (as QR codes)
- This talk is for beginners up to experts



SLIDES

The journey of a device going to production

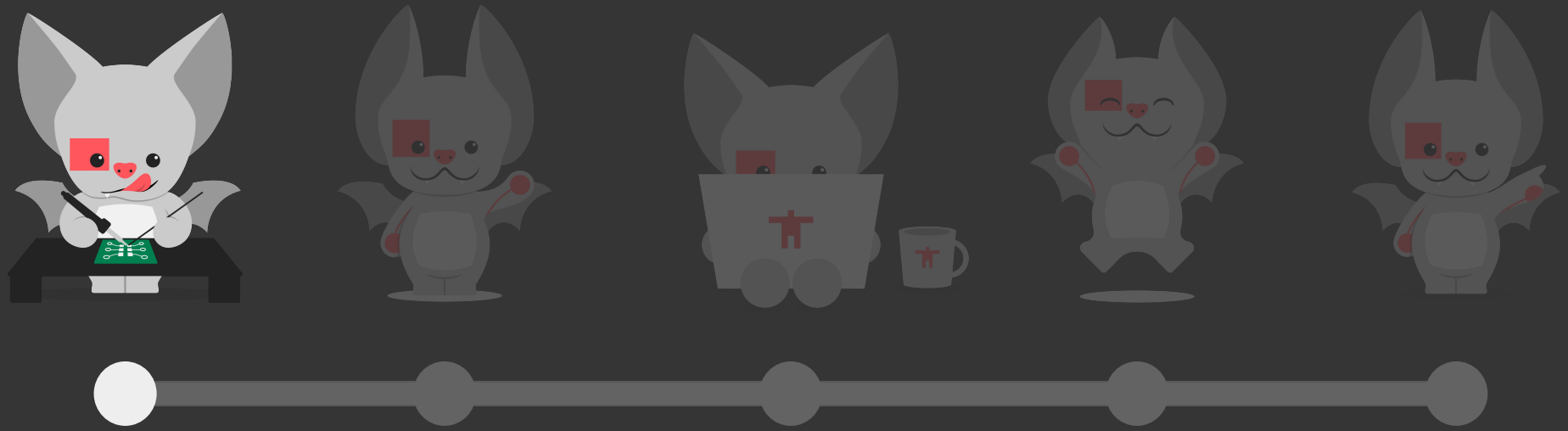
Early Prototyping On Devboards

Custom HW, First Pilot

First Devices in Production

Scaling Production

Maintaining Scaled Fleet



Early Prototyping On Devboards

- Your main goal should be validating your overall idea.
- The details of which parts you're using might not be that important, unless you are targeting a particular part family or ecosystem
- Zephyr has the benefit of being able to port between different types of hardware (silicon vendors, connectivity, etc) very quickly
- Prototyping in Zephyr has a learning curve, but then has huge benefits for future prototyping efforts once you learn

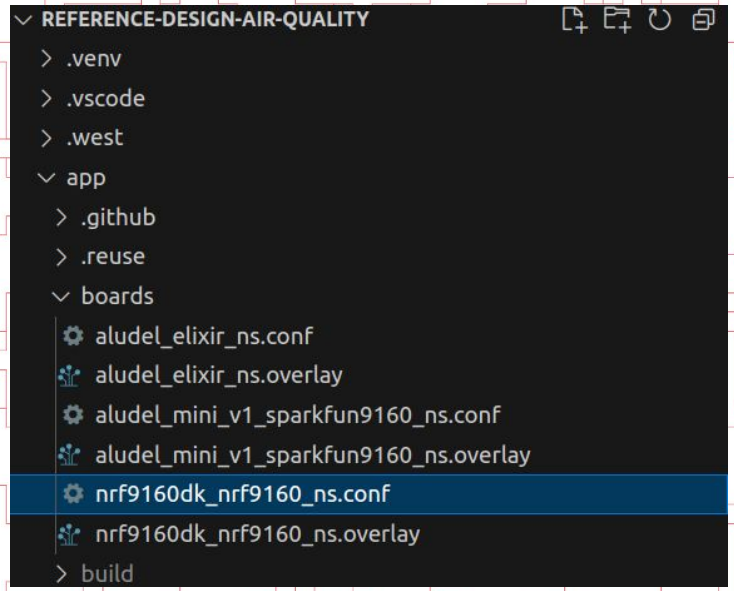
Board files

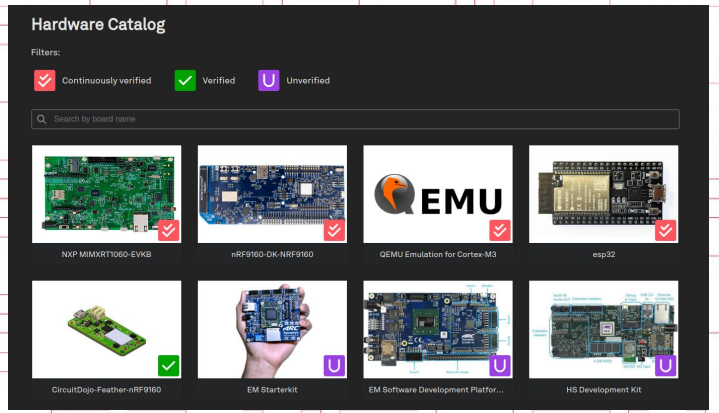
START FROM THE TOP DOWN

- One of the most powerful things in Zephyr is that vendors have encapsulated everything their development boards can do in board definitions
- This means samples are regularly tested against these boards...and you can too!
- PIece together different samples and capabilities, including connectivity, if needed.

PORTING BOARDS

- You usually won't need to port an entirely new board (that's why we're using dev boards in the first place), but looking at the process can really help you out.





Use menuconfig

UNDERSTAND ALL YOUR CONFIGURATIONS

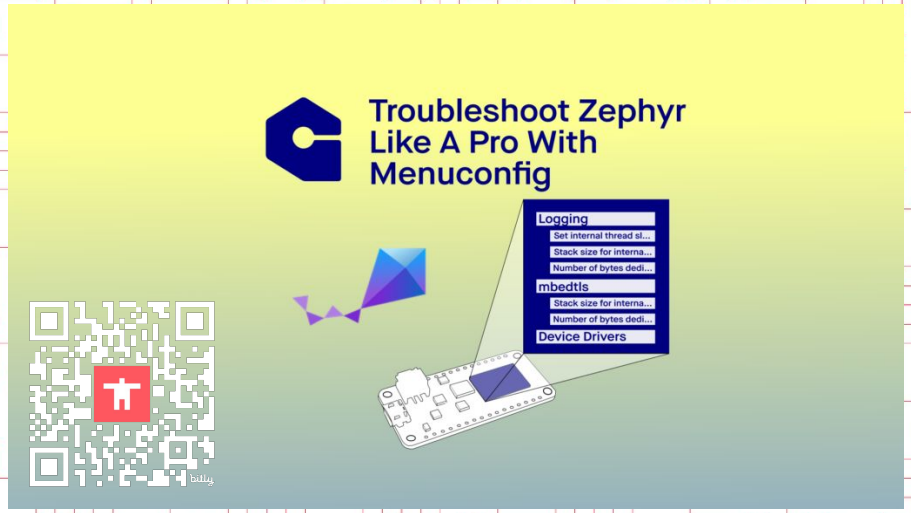
- Zephyr makes it easy to configure *very* complex systems, which can be treacherous to troubleshoot
- Menuconfig allows users to see all configurations in one place, including dependencies

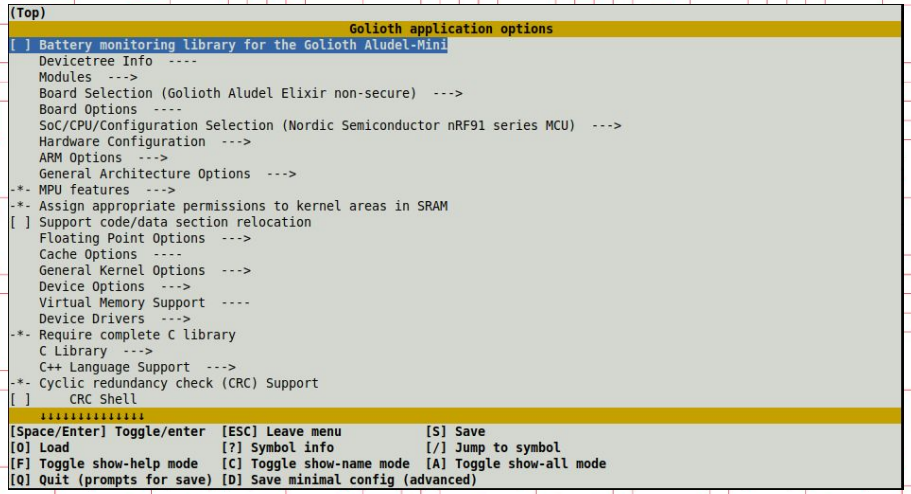
EASY TO LAUNCH, HARDER TO UNDERSTAND

- Get started with the command:

```
west build -t menuconfig
```

- Use arrow keys, space bar to navigate and select the options you want to change in your configuration.
- View and diff your `.config` files to understand which Kconfig symbols should be going into your files for future builds (to make permanent)





Sensor and I2C Shell

IMPLEMENT IN-TREE SENSORS QUICKLY

- Few development boards will have every peripheral you'll want on your final product
- Add an overlay file matching the board name you're targeting (and check the pinouts on your board) to start connecting to external sensors
- Plug in a secondary sensor and then use the sensor shell in the terminal

TRY OUT NEW SENSORS WITHOUT WRITING DRIVERS

- The above is dependent upon a driver being in-tree for the sensor you chose
- You can always "dive down the layers of abstraction" and do direct i2c reads and writes
- This could also be captive in a separate thread or work queue in order to make it non-blocking

board_name.overlay

```
&i2c0 {
      status = "okay";

      bme280@76 {
      compatible = "bosch,bme280";
      reg = <0x76>;
      };
```

The journey of a device going to production

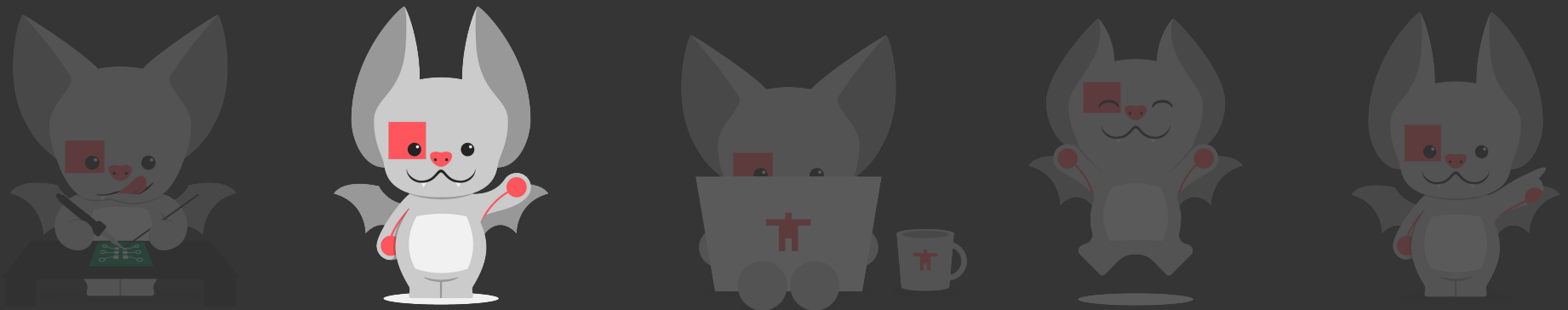
Early Prototyping On Devboards

Custom HW, First Pilot

First Devices in Production

Scaling Production

Maintaining Scaled Fleet



Custom HW, First Pilot

- First revision of custom hardware, which might not be the final form factor (my first boards are normally larger for easier debugging)
- Normally I do something like 10 units
- Enough to make sure multiple people can try out the hardware and also in case any … rapidly disassemble
- These are your test units where you figure out if your capabilities are sufficient for the idea you validated in the first stage
- You should be experimenting with the moves you'll need to take when you scale production (certificates, programming, testing)

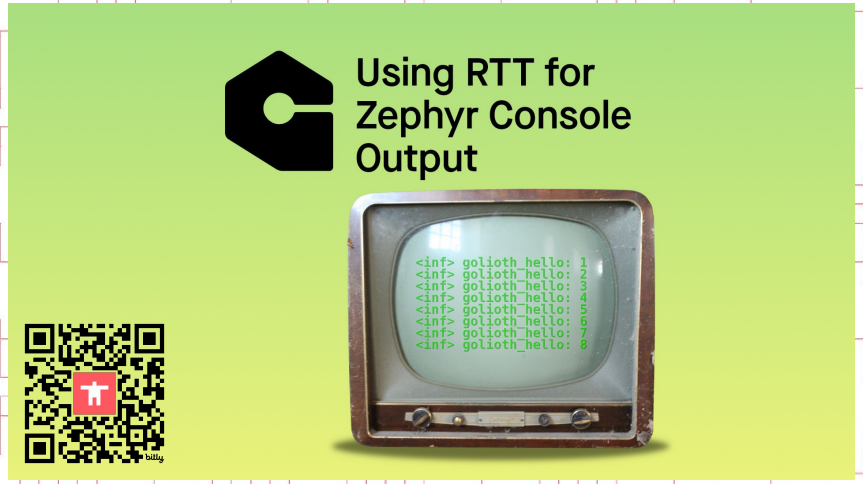
Alternative logging paths

CHANGE YOUR BACKEND TO USE RTT

- Zephyr allows you to point the logging output to multiple backends, including RTT. This becomes important if you run out of serial ports on your prototype or if you want to restrict log outputs from anything but a controlled debug header

CLOUD BACKENDS ARE POSSIBLE TOO!

- You can also mirror your backend to a service like Golioth to see the same debug data but on a remote terminal
- This will also required a network connection, which will need to be configured using Zephyr



prj.conf

```
CONFIG_UART_CONSOLE=n
CONFIG_RTT_CONSOLE=y
CONFIG_USE_SEGGER_RTT=y
```

Debugging tools

DEEP DEBUGGING CAPABILITIES ARE CRITICAL

- Since you're trying to wring out all of the bugs on this first rev of custom hardware, you need a large range of debugging tools to peer inside
- Standard ways of step debugging include GDB + VScode, and often work out of the box
- 3rd party tooling can also work with Zephyr, including Segger Ozone and SystemView. The latter allows you to analyze which tasks are consuming the most time for program execution

(Ozone) prj.conf

```
# use thread names

CONFIG_THREAD_NAME=y
CONFIG_SCHED_CPU_MASK=y
CONFIG_THREAD_ANALYZER=y
```

(SystemView) prj.conf

```
CONFIG_THREAD_NAME=y
CONFIG_SEGGER_SYSTEMVIEW=y
CONFIG_USE_SEGGER_RTT=y
CONFIG_TRACING=y
CONFIG_TRACING_BACKEND_RAM=y
```

MCUboot

INITIAL MEMORY SETUP

- Now that you're in charge of the hardware, you can start mapping the memory to suit your needs
- Starting with a built in bootloader like MCUboot ensures you have multiple update mechanisms from day 1

USING MCUBOOT

- Need to set up DTS and KConfig symbols to enable MCUboot, as well as setting up `boot_partition`, `slot0_partition`, and `slot1_partition`
- Select how to recover the bootloader, either UART or CDC ACM (requires USB)
- Signing images is supported (and suggested) and MCUboot helps to manage keys

The journey of a device going to production

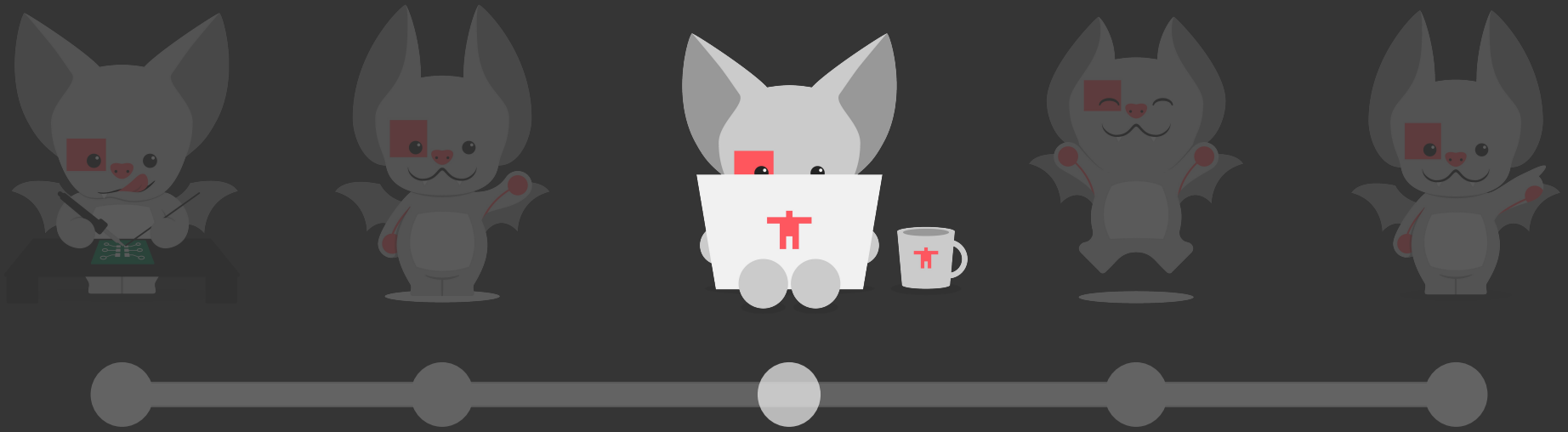
Early Prototyping On Devboards

Custom HW, First Pilot

First Devices in Production

Scaling Production

Maintaining Scaled Fleet



First Devices in Production

- Second revision of hardware
- Larger than the first run, (say 100 units) built by a contract manufacturer (CM), if you didn’t have them build the last run
- Need to be thinking about standing up testing infrastructure while also wringing out the final bugs of the product
- Also starting the cost down process and finalizing form factor

Multiple HW versions

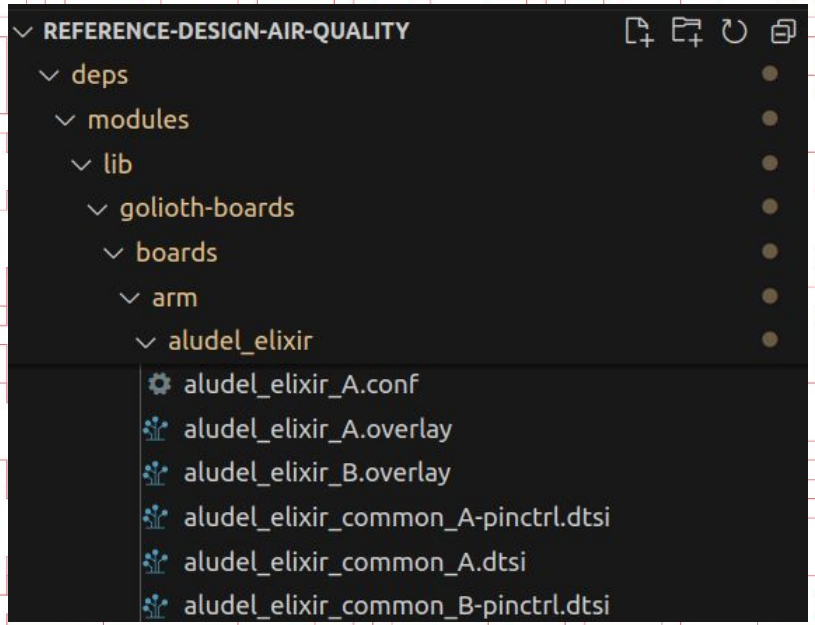
MANAGE REVS WITH BOARD FILES

- Now that you are starting to see variations in your hardware (either intentional changes to pull in new functionality or because you made a mistake), you'll want to be able to target each rev on the command line when building.
- There are defaults that are baked in (rev B on the current version of the Elixir)


```
# Build firmware for Rev A
west build -b aludel_elixir_ns@A

# Build firmware for Rev B
west build -b aludel_elixir_ns@B

# Build firmware for the "default"
revision (which is currently Rev B)
west build -b aludel_elixir_ns
```



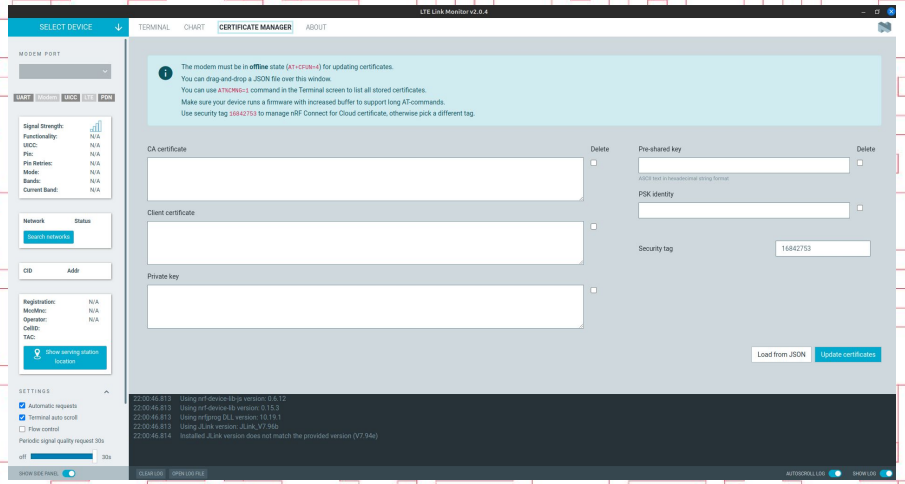
Provisioning

CREATE AND USING CERTIFICATES

- As devices roll off the assembly line, they can be granted individual device certificates signed using a trusted chain of root certificates and intermediate certificates
- This is more secure than “Pre-Shared Keys” (PSK) which can be extracted from devices and utilized by a bad actor
- When they first connect, the certificates are verified against the chain of trust and a record of the trust-verified device is created. This simplifies the registration of a large influx of new devices, as happens in a production environment

LOADING CERTIFICATES

- Common ways of loading certificates includes having a primary firmware on the production line that is only for loading certificates into the modem or using a vendor tool like nRF Connect for Desktop (right)



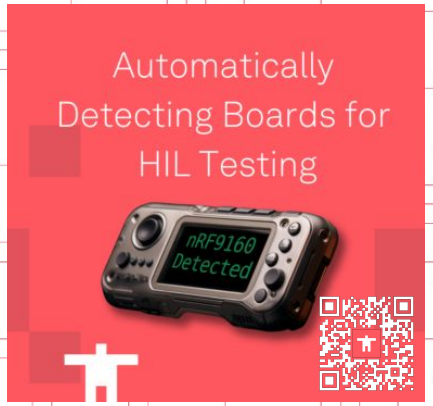
Testing Infrastructure

BUILD ON TOP OF ZEPHYR INFRASTRUCTURE

- Arguable that this should start even sooner in the process, but at this volume of products (100+), automation is necessary
- Utilize a wide range of testing infrastructure built into Zephyr for internal (ecosystem) testing, but for your specific device
  - Pytest
  - Twister
  - Devicesim

HARDWARE-IN-THE-LOOP

- You can push individual builds out via GitHub actions to hardware units (self-hosted runners).
- The output of these tests can be tested via pytest and reported in bulk



Remote FW Update

WORKING IN CONJUNCTION WITH MCUBOOT

- Even at 100 units, it becomes unmaintainable to plug in and update 100 units using physical means like a USB cable
- Create firmware updates that are not the full image (`app_update.bin`, instead of `merged.hex`)
- Download an image from a remote server, verify the nature of that image, and push it into the alternate slot on mcuboot
- Call `mcuboot` from the uart shell to see available slots

Execute `mcuboot` shell command in Zephyr to confirm that new firmware is running from primary area (first application slot):

```
uart:~$ mcuboot
swap type: none
confirmed: 1

primary area (1):
  version: 1.2.3+0
  image size: 221104
  image hash: 40710f0bd8171d7614b13da4821da57066f4431e4f3ebb473de9e95f6467ae65
  magic: good
  swap type: test
  copy done: set
  image ok: set

secondary area (2):
  version: 0.0.0+0
  image size: 221104
  image hash: f48973eed40a9d30795df7121183e7a828e9b89aa5ee84f2db1318f7cf51be0b
  magic: unset
  swap type: none
  copy done: unset
  image ok: unset
```

The journey of a device going to production

Early Prototyping On Devboards

Custom HW, First Pilot

First Devices in Production

Scaling Production

Maintaining Scaled Fleet

Scaling Production

- You have received and tested your second revision, maybe you're on a 3rd revision if you’re scaling board to a final form factor
- This is the stage where you order lots and lots of devices (say 1,000)
- As your sales expand, it’s also likely that you need to account that the hardware is going to different locations around the world
- Testing at the factory needs to be automated and without issues

Optimize Battery

TURN IT OFF WITH PINCTRL

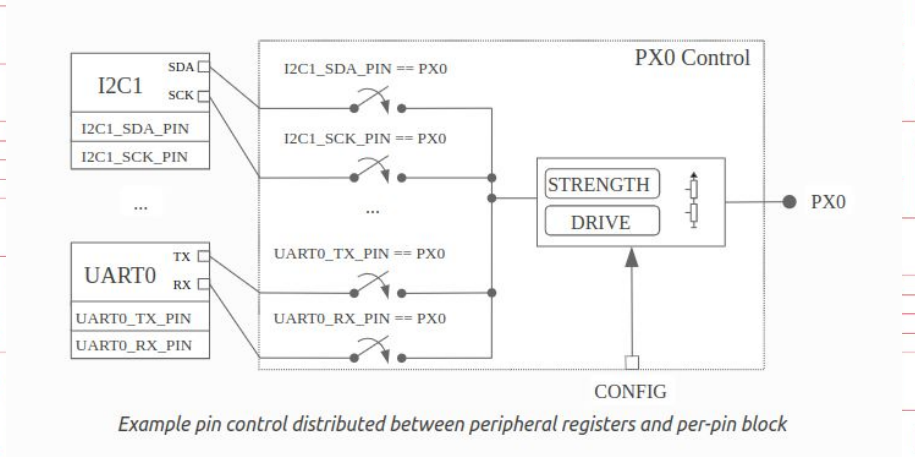
- Set different pins into normal and low-power states to save on pullups / drive strength leakage

PM SUBSYSTEM

- Power management (PM) subsystem taps into the capabilities of the silicon to reduce power. Turn off anything you're not using (in Devicetree) and then put the part into lower power mode using PM
- Can set power domains to target different sections of a board and allow certain aspects to wake up the chip from a low power state

POWER DOWN WITH REGULATOR SUBSYSTEM

- Other ways to save power are shutting down power domains. These can be enabled at boot and selectively powered down during operation
- We do this with the 3V3 sensor subdomain on the Elixir to power down secondary modems and a range of sensors



Example pin control distributed between peripheral registers and per-pin block

Optimize Security

SECURE YOUR BOOTLOADER

- Create a signature key to make sure only images created by you can be loaded. This is relevant for local firmware loading (using a programmer) as well as remote loading of firmware (OTA)

TRUSTED FIRMWARE (TF-M)

- Using this feature will depend on the hardware capabilities and having secured storage areas on the silicon.
- Two links are a talk by Kevin Townsend and some of Nordic's resources for TF-M built on top of the Zephyr cryptographic examples.

UTILIZE PKI

- Private Key Infrastructure (PKI) helps you to maintain your certificates and keys. You definitely don't want someone accessing your cryptographic assets and losing them is almost as damaging.

Optimize Data

SERIALIZATION

- Zephyr has a built-in protobuf implementation and a subsystem that can handle others
- Golioth uses JSON and CBOR encoding and is available as a small switch to the API calls from the device

SEND THE MOST IMPORTANT MESSAGES

- As your fleet continues to stabilize, you should require fewer updates from each device unless there is a problem situation
- Individual modules can be turned off from within the program
- Golioth can send a Remote Procedure Call (RPC) to trigger a script that cycles through each submodule and turn down the logging level for each.

```c
static enum golioth_rpc_status on_set_log_level(QCBORDecodeContext *request_params_array,
                                                QCBOREncodeContext *response_detail_map,
                                                void *callback_arg)
{
    double a;
    uint32_t log_level;
    QCBORError qerr;

    QCBORDecode_GetDouble(request_params_array, &a);
    qerr = QCBORDecode_GetError(request_params_array);
    if (qerr != QCBOR_SUCCESS) {
        LOG_ERR("Failed to decode array item: %d (%s)", qerr, qcbor_err_to_str(qerr));
        return GOLIOTH_RPC_INVALID_ARGUMENT;
    }

    log_level = (uint32_t)a;

    if ((log_level < 0) || (log_level > LOG_LEVEL_DBG)) {

        LOG_ERR("Requested log level is out of bounds: %d", log_level);
        return GOLIOTH_RPC_INVALID_ARGUMENT;
    }

    int source_id = 0;
    char *source_name;

    while (1) {
        source_name = (char *)log_source_name_get(0, source_id);
        if (source_name == NULL) {
            break;
        }

        LOG_WRN("Settings %s log level to: %d", source_name, log_level);
        log_filter_set(NULL, 0, source_id, log_level);
        ++source_id;
    }

    return GOLIOTH_RPC_OK;
}
```

Optimize Test Stands

CREATING TEST STANDS IS AN ART

- Creating effective test stands could be another topic unto itself
- The best test stands program and run the minimum amount of tests that properly cover all characteristics of a device

FASTER TESTING WITH CUSTOM SHELL COMMANDS

- While shell commands don't need to be exclusive to a test stand (useful for many things!), they can also be used to exercise many parts of a design (think: LEDs, outputs, RF power tests, more)
- This would likely be test firmware that is not the same firmware sent to the field, and the final step would be programming the production firmware onto the board

The journey of a device going to production

Early Prototyping On Devboards

Custom HW, First Pilot

First Devices in Production

Scaling Production

Maintaining Scaled Fleet

Maintaining Scaled Fleet

- Fleets continue to grow as additional CM capacity comes online (say 10,000+ devices in a fleet)
- Automation is increasingly important as your fleet grows, because checking on each unit becomes untenable
- Future hardware revisions are possible, but continuing to revise the firmware of an existing fleet requires firmware change management

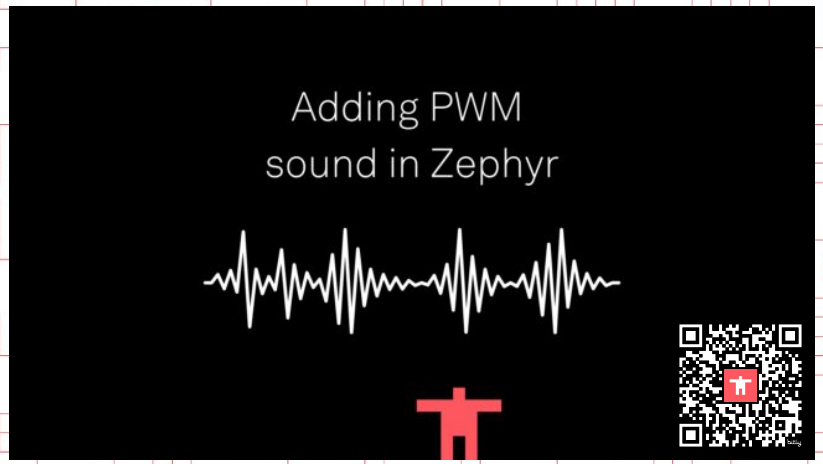
Command & Control

UTILIZING SETTINGS

- The settings subsystem allows you to create device settings in non-volatile storage and boot up with the setting in tact
- Golioth has a service that ties directly into this subsystem and synchronizes/verifies delivery of the setting from the cloud

UTILIZE THREADS FOR TRIGGERABLE COMMANDS

- Using Zephyr's Threads and Work Queues allows you to portion up work to be done at the scheduler's leisure
- This is a great model for interacting with Remote Procedure Calls (RPCs) on Golioth
- One example is triggering a song to play on a PWM buzzer using RPCs. Once the external trigger (RPC) is called, it kicks of a thread to pulse the PWM and drive a speaker with notes



Tracking volume issues

EXCEPTION REPORTING VIA DEVICE MANAGEMENT

- At this scale (10K+ devices), you can only really deal with the outlier units, based on good reporting
- Cloud side automation is often key in order to watch for bad behavior
- Visualization tools can help you to take quick looks at behavior across a range of devices.

WATCHDOGS

- Servicing watchdogs (left link) will ensure your devices respond within a certain timeframe or reset the device and try again.

UTILIZING TRACES

- Tracking issues during crashes via core dumps (right link) helps to diagnose wide issues. These can work with services like Memfault, work well with Zephyr (installable via library).

Extensibility

FILE SYSTEM

- Using the easy-to-use fatfs or littlefs allows users to push arbitrary files into the device. This is useful for things like visualizations (using LLVM) or algorithm tunings

LLEXT

- From Ben Cabe's newsletter (subscribe!)
  - Linkable Loadable Extensions (llext) subsystem makes it possible to dynamically extend the functionality of an application at runtime.
- This means you could have one base level firmware and a bunch of LLEXTs that create different "flavors" of your hardware/firmware.

LOADABLE OVER OTA

- Pushing binaries to a device via blockwise transfer can enable both file system and LLEXT extesion

The journey of a device going to production

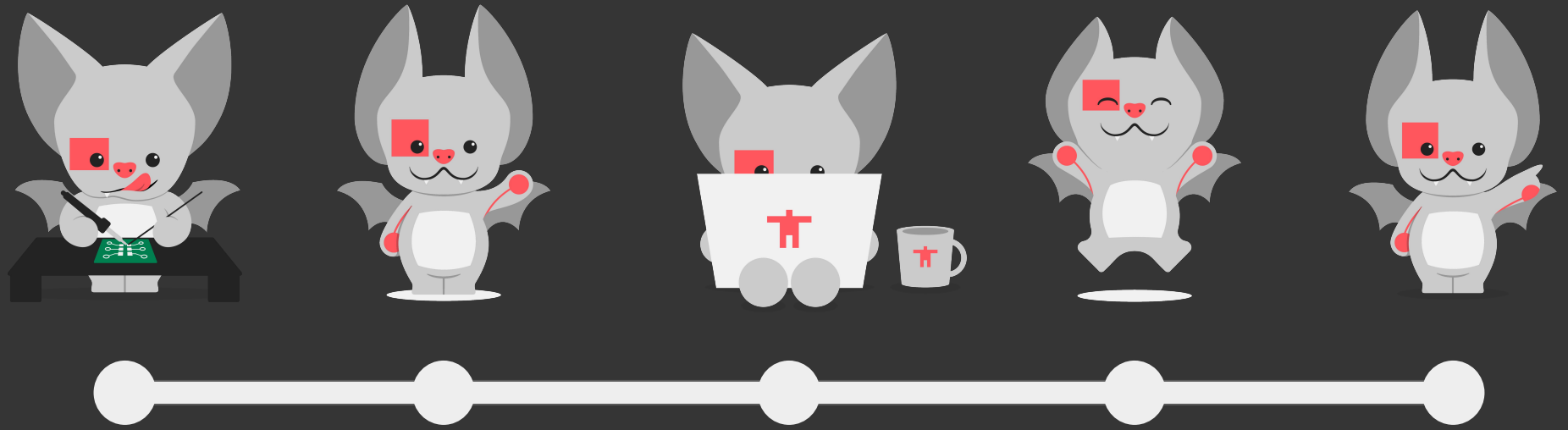
Early Prototyping On Devboards

Custom HW, First Pilot

First Devices in Production

Scaling Production

Maintaining Scaled Fleet



Go build something big

golioth

SLIDES