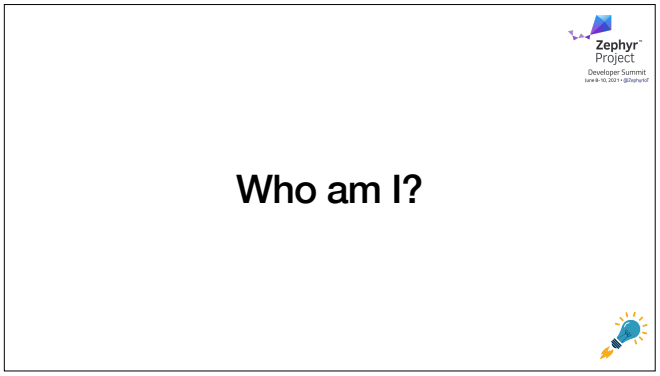
Thanks for joining my presentation on Developing Hardware for Zephyr. I've very honored to be here. Thanks to everyone on the Zephyr team who have put together this awesome vent!

Let's get this thing started!

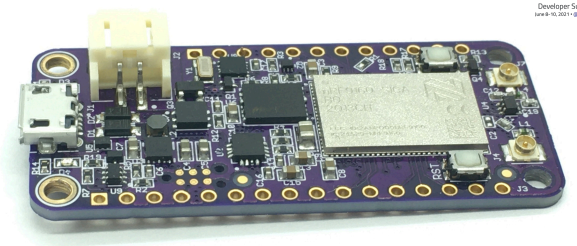


First, a little background on who I am..

I'm Jared. I'm the owner of Circuit Dojo and the sole developer of the nRF9160 Feather among other things! In a previous life I cut my teeth in Silicon Valley developing hardware for startups and large companies alike.

What got me into trouble.. 😅

I now focus on helping my clients make their products a reality. Along with my consulting I stumbled my way into a happy accident that has been keeping me quite busy…



This is the nRF9160 Feather. I designed it from the ground up with the use cases that were most interesting to me. It has some crazy low power shutdown modes, dual U.FL connectors for external GPS and LTE antennas, onboard storage and more.

The picture you're looking at right now is one that I've personally assembled. The board is a 4 layer OSH Park board and most of the parts were procured through Digikey at the time.

Prototyping 🔨

I won't go into much depth in term of prototyping the nRF9160 Feather. I will share some pictures from the process though..

The close up shot of the board early is likely one of these boards right here. Turns out it was harder to solder the nRF9160 than expected. It's all about a steady hand, correct solder paste placement and also having the correct solder-mask and stencil openings.

(Shots are from my electronics lab in my home office)

Diving into Device Tree 🌳

Now, we've got some prototype hardware. But what are we going to do with it?

Zephyr has some really great built-in samples and examples, but how do we get this previously non-existent board working with Zephyr?

The first place to look is how Zephyr defines hardware. And all of that action happens in the device tree.

Why Device Tree?

- Unified board definition syntax
- Samples "just work" when using standardized interfaces
- Makes it easier to get people working on your hardware

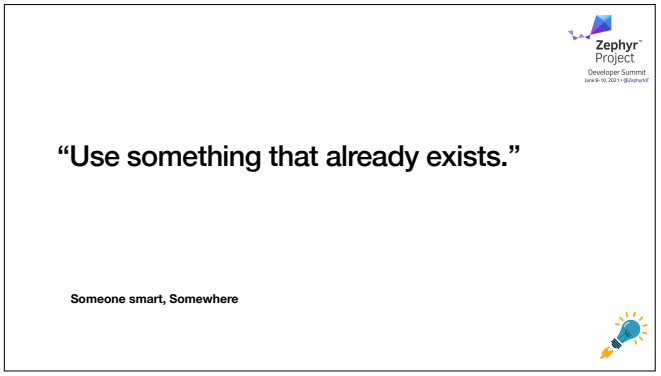
Zephyr allows you to define all the functionality of your board in a standardized way. That way you can use your board with many of the included Zephyr samples and they "just work."

Most of the time, device tree files are stored within the **boards** directory. The boards directory is sorted by architecture. So make sure you're putting things in the right place!

When you finish developing your device tree entry, you'll find that things "just work" and there's not much lower level hardware tweaking required to talk to

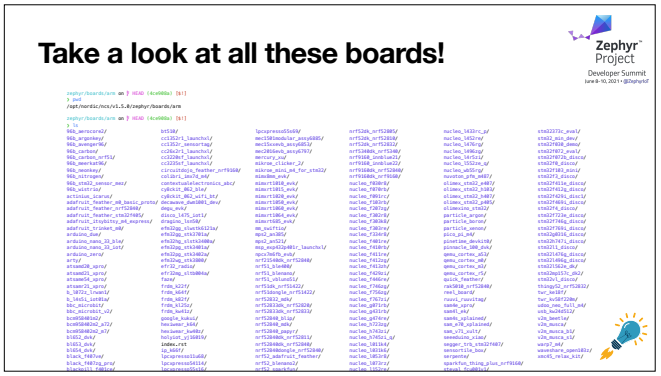
an accelerometer or other sensors. Caveat being here that Zephyr has support for the device you'd like to integrate with!

Developing and publishing your device tree definitions also makes it a ton easier for fellow Zephyr developers to use your board. And if you're all about making it easy for developers, this should be a no brainer!



One way that you can make your life a ton easier is by basing your device tree entries on ones that already exist.

For the nRF9160 Feather, I used the nRF9160DK as a reference to create all the device tree files necessary to get the nRF9160 Feather working.



Here's the output of **ls** within the **zephyr/boards/arm** directory.

Now worries about counting the entries. There are 230 in this screenshot. And this number is **growing every day**.

So now that you know that *everyone is doing it,* let me peer pressure you into creating your own board definitions and publishing them to the Zephyr repository.

Now let's look into the details and makeup of a device definition in Zephyr..

Basic device tree structure

```
boards/arm/nrf52dk_nrf52832 on
 HEAD (4ce908a) [$!]
❯ ls
Kconfig
Kconfig.defconfig
doc/
nrf52dk_nrf52832.yaml
Kconfig.board
board.cmake
nrf52dk_nrf52832.dts
nrf52dk_nrf52832_defconfig
```

- Board level Kconfig
- Board level Kconfig defaults/selects
- Board level .yaml binding
- _defconfig
- .dts (most of the magic happens here! 🧙)

So, let's use the nRF52-DK as an example of the barebones requirements for device tree development. Let me hit on some of the most important files.

There's a device Kconfig where you can create configuration variables that you can use later on in code.

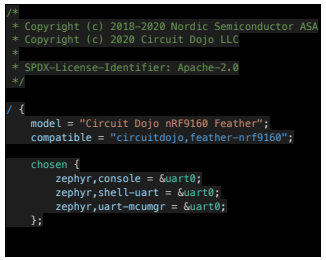
nrf52832_mdk_defconfig is similar to your projects **prj.conf** where you can set configuration variables that will propagate to your applications. This is very handy for enabling features you're *always* going to use no matter what.

There's also the .defconfig file which allows you to ***select*** or enable Zephyr features automatically when using this board.

The **doc/** folder includes the restructured text documentation for your board. I recommend you check out the documentation yourself to see how other projects have formatted theirs.

Finally, the star of the show, we have the .dts file where most of your board definition will live. More complicated boards, like the nRF9160 Feather, require multiple board definitions but that's something for another talk.

Pre-define your interfaces



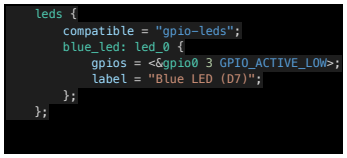
- Create a root node for the device itself
- Define your console, shell and mcumgr output

Make sure you give the originating project credit! Nordic gets 100% credit here.

Compatible defines the root device type which is defined in the binding for the device itself within the device folder (usually ending in .yaml)

The chosen section helps your define the built-in zephyr consoles etc and assign a specific interface. In this case we're using uart0 for the console, shell and mcumgr.

Pre-define your interfaces ..



- Lets define a LED node.
- blue_led defines the blue LED on the nRF9160 Feather
- Pin 0.03. Active Low

Here's an example of the leds node. This node allows you to define all the leds on your board. The nRF9160 Feather has only one LED but if you, say, had a red led, you could easily create a **red_led** sub-node.

For **blue_led**, we need to point to a GPIO interface, pin number and active state (active high or active low). You can also provide a device label for accessing this device directly in your code.

We'll also talk about connecting LED instances as an alias so they can be used in Zephyr's standard examples shortly!

Again the **compatible property** references the device binding to use. These are located in **zephyr/dts/bindings** if you ever need inspiration if you ever need to create your own device bindings for custom peripherals.

Pre-define your interfaces more..

```
buttons {
    compatible = "gpio-keys";
    button0: button_0 {
        gpios = <&gpio0 12 (GPIO_PULL_UP |
            GPIO_ACTIVE_LOW)>;
        label = "Switch 1";
    };
};
```

- Lets define a button!
  - *Very* similar to an LED definition

Similarly, here's a button definition. You see it has many of the same components an LED has except it is using a different binding.

Using **gpio-keys** instead of **gpio-leds.**

Also including modifiers for the GPIO like including a pull-up along with how it's configure (active low/high)

Pre-define your interfaces more…

```
/* These aliases are provided for
compatibility with samples */
aliases {
    led0 = &blue_led;
    bootloader-led0 = &blue_led;
    pwm-led0 = &pwm_led0;
    sw0 = &button0;
};
```

- Aliases
  - Rename your peripherals to match common examples

Also in the same area of the device definition, you. May notice an *aliases* node. This node helps you rename or give a node an *alias* for certain examples.

For example, led0 is used across Zephyr as the main LED in most projects.

bootloader-led0 is used in MCUBoot as the default LED to turn on while MCUBoot is enabled.

As you explore Zephyr more, you'll undoubtedly need to define a new alias. This is the node where you can make that happen!

As a sidenote, you generally do not want to modify your device tree files within Zephyr especially for a one-off change. That's what device .overlay files are for. I'll get into .overlay files briefly in a second!

Pre-define your interfaces even more…..

```
&i2c1 {
    compatible = "nordic,nrf-twim";
    status = "okay";
    sda-pin = <26>;
    scl-pin = <27>;

    pcf85063a@51 {
            compatible = "nxp,pcf85063a";
            label = "PCF85063A";
            reg = <0x51>;
    };
};
```

- Define your peripherals
  - SPI
  - I2C
  - UART
  - Etc

Referencing Nordic's device binding. This helps prepare Zephyr to use Nordic's specific TWIM (I2C) driver.

In some of the cases for pins, you need to add an offset of 31 for every Port you go above Port 0.

Built in low power RTC that is onboard the nRF9160 Feather. The pcf85063a is an example of a device that Zephyr doesn't have 100% support for. I had to create a supplementary driver using the timer API.

Test and share your board definitions ❤️

Once you're comfortable with your device definitions, it's time to test them!

Test em

- Blinky
- Button
- LittleFS
- AT Client

There are a bunch of samples that I used to test the nRF910 Feather including:

Blinky to test the onboard LED connected to Digital output 7
Button to test the onboard Switch
The LittleFS to test out external flash and to give LittleFS a spin. (Located in **zephyr/samples/subsys/fs/littlefs**)
The AT Client example from nRF Connect SDK to query the nRF9160 and connect to an LTE network.

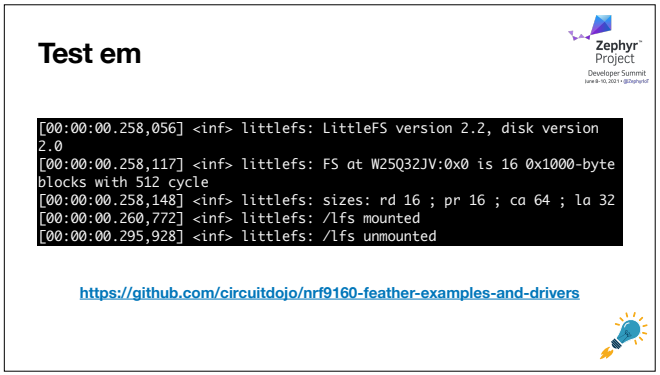
Test em

You'll notice in the the LittleFS sample that I've contributed the .conf and .overlay for the nRF9160 Feather. The main point of of these files are to override device specific definitions that you've created within the **zephyr/boards/arm/yourboard** directory.

For example, if you need to create a new alias, you can do so right within the **circuitdojo_feather_nrf9160.overlay** file. Think of it like a project specific .dts diff. In the case of this example, the overlay is used to remap some storage partition information to the external SPI flash.

The .conf files are supplementary to **prj.conf.** That way you can use multiple targets for the same sample. Any hardware differences can be handled within the .conf and .overlay.

Test em



Here's an example output from the LittleFS example properly working!

You can checkout some of the other samples and drivers that I've put together for the nRF9160 Feather at the link on this page. You can also search on Github for **nRF9160 Feather Examples and Drivers**. It should be one of the first results!

Submit your PR!

Example commit message:

boards: arm: nrf: Adding updated Circuit Dojo board definitions.
Adding the Circuit Dojo nRF9160 Thing Plus device tree definitions.
Both secure and non secure targets.

Adding LIS2DH to nRF9160 Feather
definitions. Updating flash to W25Q32JV.

Signed-off-by: Jared Wolff <hello@jaredwolff.com>

Once you're happy with your work, it's time to submit a pull-request to get it merge with Zephyr!

The best way to do this is fork Zephyr and then commit your changes to the latest master branch.

This is an example of a commit message from me updating some of the board definitions for the nRF9160 Feather. Every commit includes the area that you're adding to, a summary and then a longer description. The "Signed-off-by" at the end is also necessary.

Make sure you only include your .conf, .dts and .rst files only. No license files necessary.

For Nordic based projects, it takes some time but eventually Nordic does merge the changes from top of tree into their branch for the use in nRF Connect Studio. In the meantime you can always copy the folder from your **zephyr/boards/arm/board name** folder to your working copy of NCS as a stopgap. Or import it in

---

Wait for it..

```
# "projects" is a list of git repositories which make up the NCS
# source code.
projects:
  - name: nrf9160-feather-board-definitions
      remote: circuitdojo
      revision: v1.3.x
      path: zephyr/boards/arm/circuitdojo_feather_nrf9160/
      import: true
```

You don't have to wait, just import it into **west.yml** and run **west update**

For Nordic based projects, it takes some time but eventually Nordic does merge the changes from top of tree into their branch for the use in nRF Connect Studio. In the meantime you can always copy the folder from your **zephyr/boards/arm/board name** folder to your working copy of NCS as a stopgap. Or import it into your **nrf/west.yml** and then run **west update**.

Thank you for taking the time to check out my presentation! I hope you've learned a little bit about building your own hardware and getting it running on Zephyr.

Slides are available at the address I have here. Everyone who signs up today also gets a hefty discount on everything nRF9160 Feather.

You can also check out my website (jaredwolff.com), I've written a ton of articles about Zephyr and the nRF9160. I hope you find it useful for your own hardware projects.

Thanks again!