An Introduction to Zephyr Device Driver

... how to handle my own driver

5. June 2026

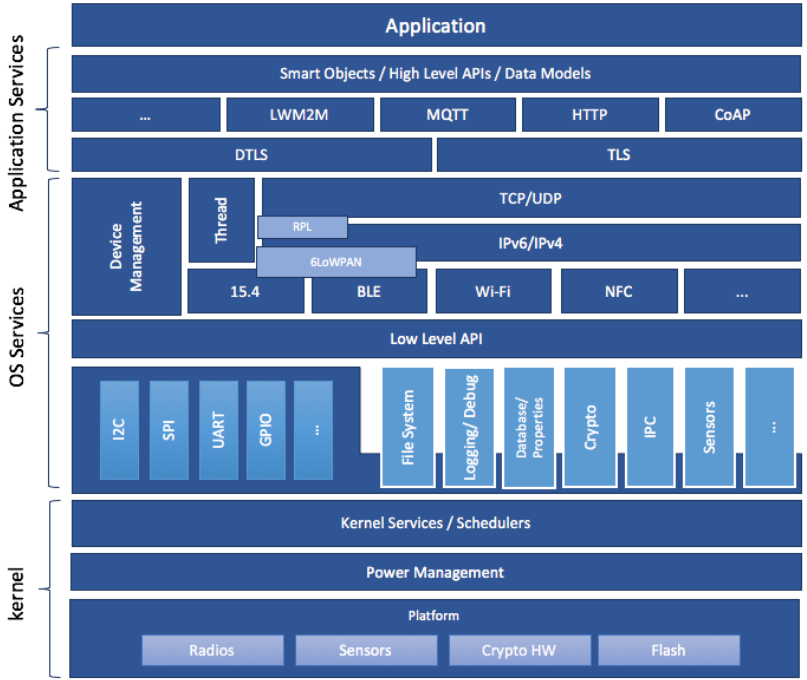
Christian Kurz

Christian.Kurz@nordicsemi.no

Zephyr System Architecture

Zephyr‘s Drivers:

- Generic Peripheral Drivers
- 220+ Sensors Already Integrated [1]
- Crypto Drivers
- File System
- Debug/Logging Drivers
- Networking Drivers
- Communication Protocl Stacks (e.g. Bluetooth LE, Wi-Fi, IEEE802.15.4, OpenThread, NFC, …)

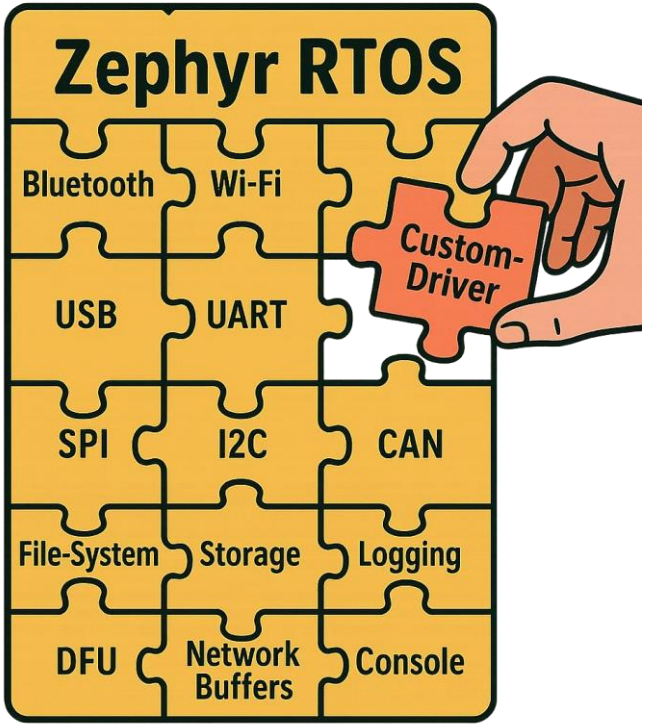


[1] See https://www.zephyrproject.org/wp-content/uploads/2025/01/Zephyr-Overview-20250113.pdf

How to add a custom Driver?

- Where should I store the driver?

  Or how can I make my Driver part of the Zephyr/nRF Connect SDK Code Base?



Zephyr Application Types

Repository	Workspace	Freestanding
An app located in Zephyr repository is referred to as a *Zephyr Repository Application* (e.g. hello_world sample)	An app located within a workspace but outside the Zephyr repository is referred to as a *Zephyr Workspace Application*	*Zephyr Freestanding Application* is an app located outside a Zephyr workspace (e.g. nRF Connect SDK installation)

```
zephyrproject/
├── .west/
│   └── config
└── zephyr/
    ├── arch/
    ├── boards/
    ├── cmake/
    ├── drivers/
    ├── samples/
    │   ├── hello_world/
    │   └── ...
    ├── tests/
    └── ...
```

```
zephyrproject/
├── .west/
│   └── config
├── zephyr/
├── bootloader/
├── modules/
├── nrf/
├── <vendor/private-repos>/
└── applications/
    └── app/
```

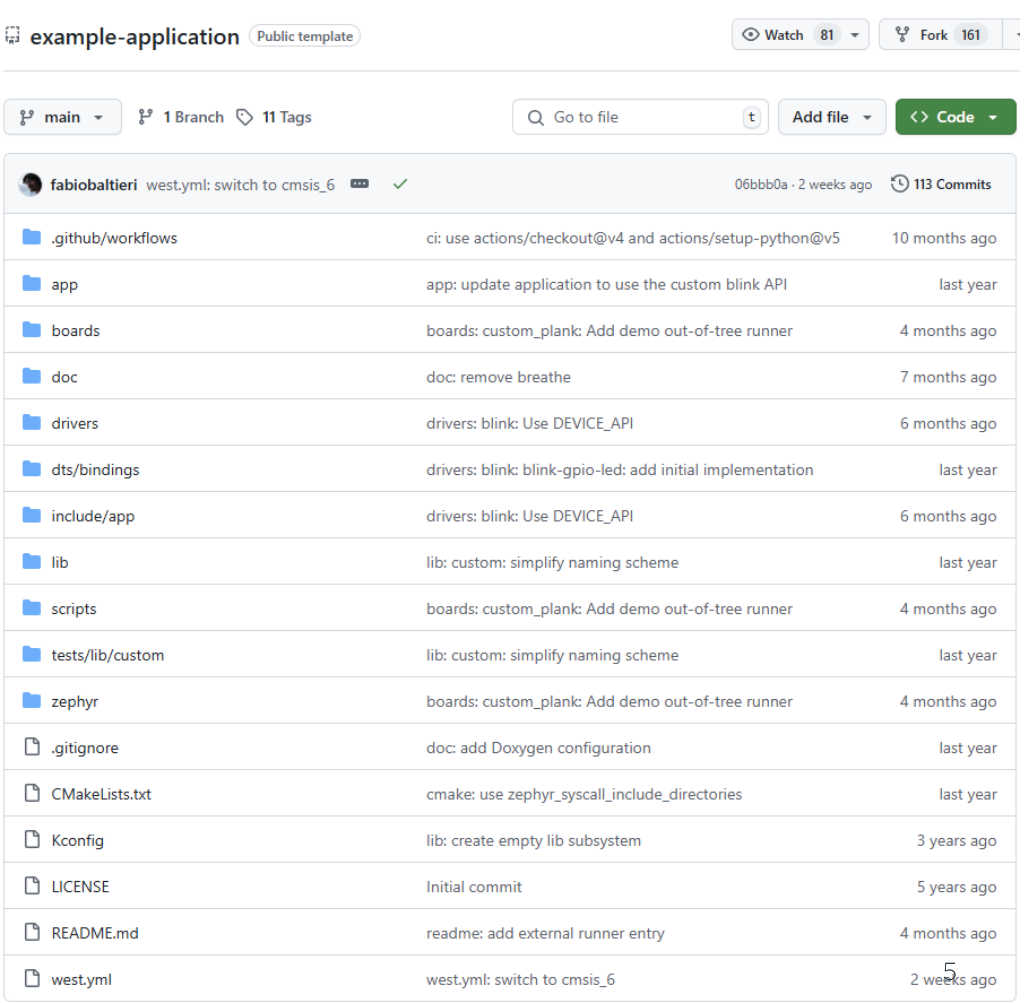
```
<home>/
├── zephyrproject/
│   ├── .west/
│   │   └── config
│   ├── zephyr/
│   ├── bootloader/
│   ├── modules/
│   └── ...
├── MyDrivers
└── app/
    ├── CMakeLists.txt
    ├── prj.conf
    └── src/
        └── main.c
```

Adding custom Driver as a *west project*

https://github.com/zephyrproject-rtos/example-application

or https://github.com/nrfconnect/ncs-example-application

- Reference on how to structure Zephyr-based applications:
  - app → Application folder (custom project)
  - boards → custom Boards
  - drivers → custom drivers
  - dts/bindings → custom binding files
  - lib → custom libraries



Adding custom Driver as a *Module*

- Custom Driver File Structure

```
C:\Nordic\MyDrivers>tree /f
C:.
    CMakeLists.txt
    Kconfig

    drivers
        CMakeLists.txt
        Kconfig

        my_led
            CMakeLists.txt
            Kconfig
            my_led.c

    dts
        bindings
            my_led
                nordic,my_led.txt

    include
        my_led
            my_led.h

    zephyr
        module.yaml
```

- Adding custom driver as a „Module“
  → needed file: *zephy/module.yaml*

- Zephyr build system is looking for it and checks *CMakeLists.txt* and *Kconfig* file in C:/Nordic/MyDrivers folder which point to additional files stored in sub-folders

Adding custom Driver as a *Module*

The user project must include this module:

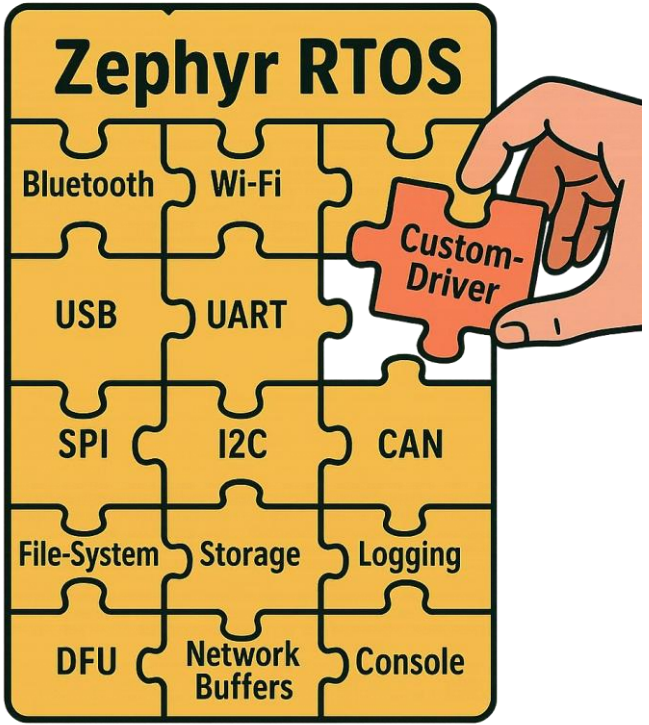
hello_world > M CMakeLists.txt

```
1   # SPDX-License-Identifier: Apache-2.0
2
3   cmake_minimum_required(VERSION 3.20.0)
4
5   # Add additional module
6   set(ZEPHYR_EXTRA_MODULES "C:/Nordic/MyDrivers")
7
8   find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
9   project(hello_world)
10
11  target_sources(app PRIVATE src/main.c)
```

How to add a custom Driver?

- Where should I store the driver?
- How to add the Driver C Code to my project?

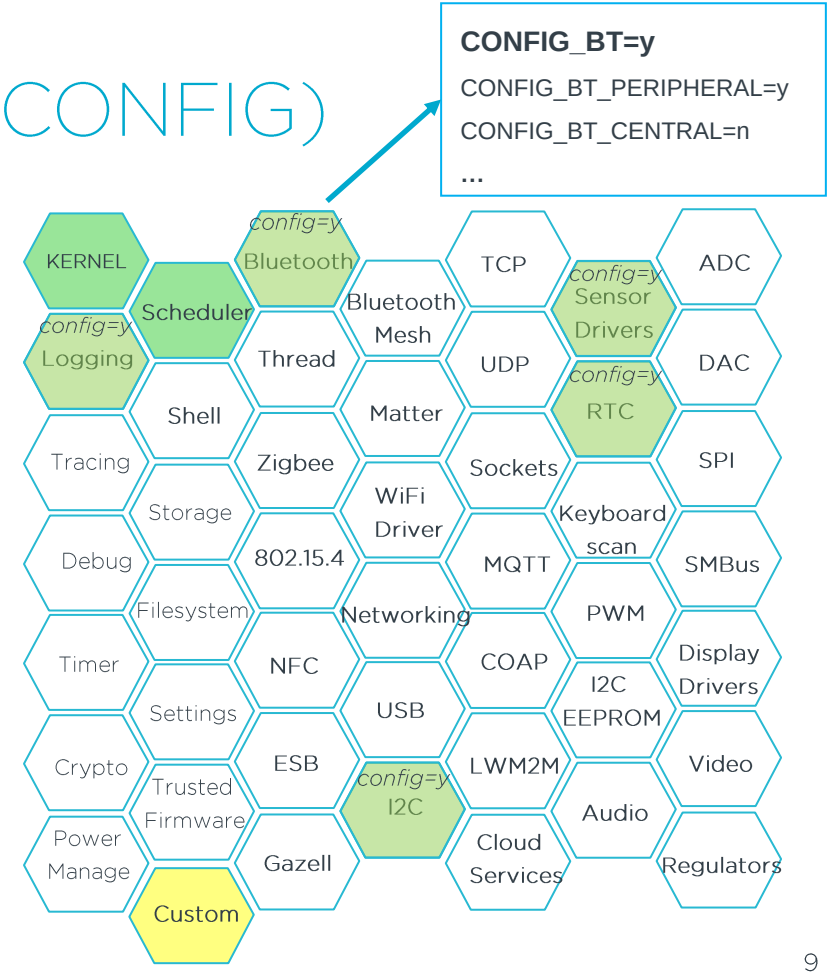
The Driver is now part of the Code Base, but how to add it to my project?



Configuration System (KCONFIG)

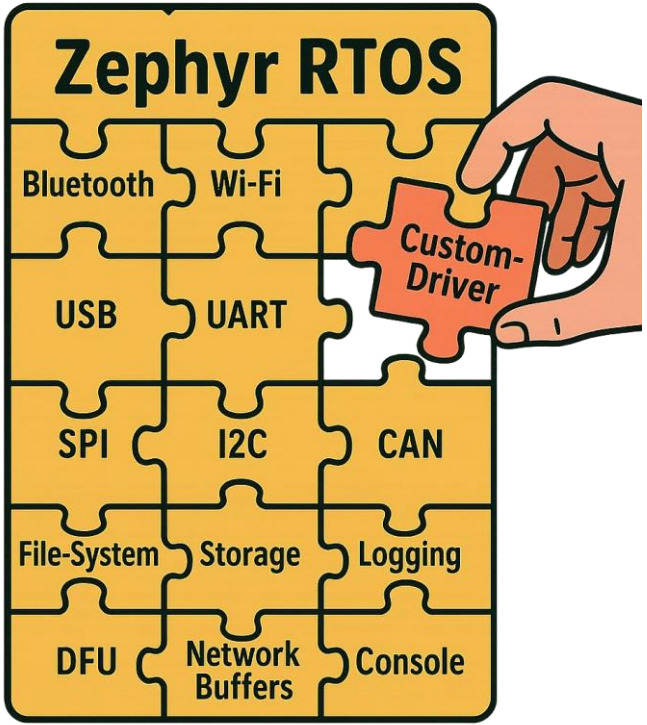
- Zephyr Kernel and Sub-System can be configure at build time
  - Add software modules to your project
  - If a software module was added, further CONFIG symbols appear and allow you to configure the module
- Custom Kconfig is possible

- Goal: Configure software features without changing source code



How to add a custom Driver?

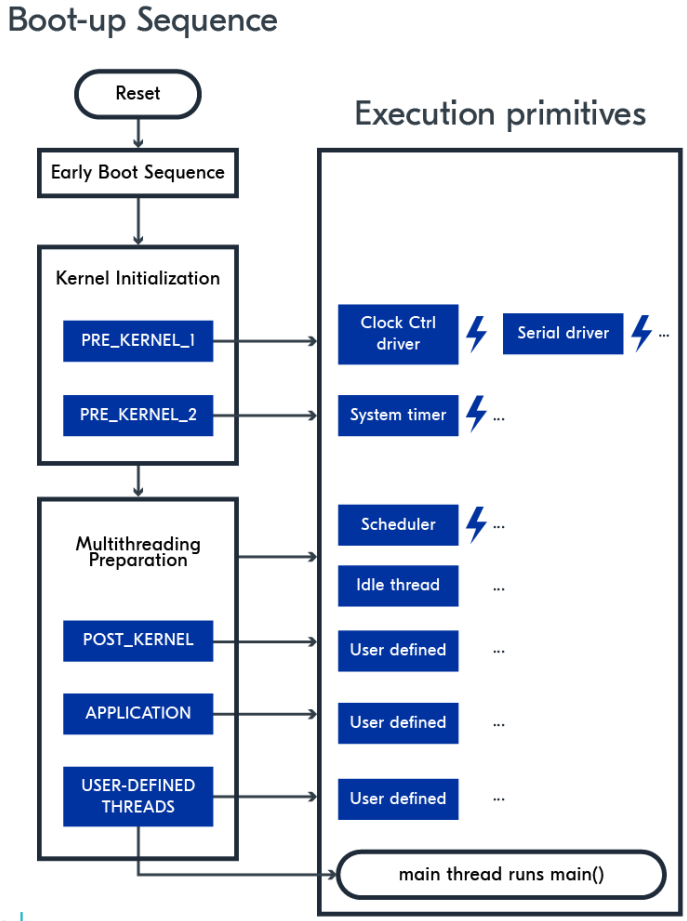
- Where should I store the driver?
- How to add the Driver C Code to my project?
- How to Initialize/Start the Driver



Zephyr's Boot-up Sequence

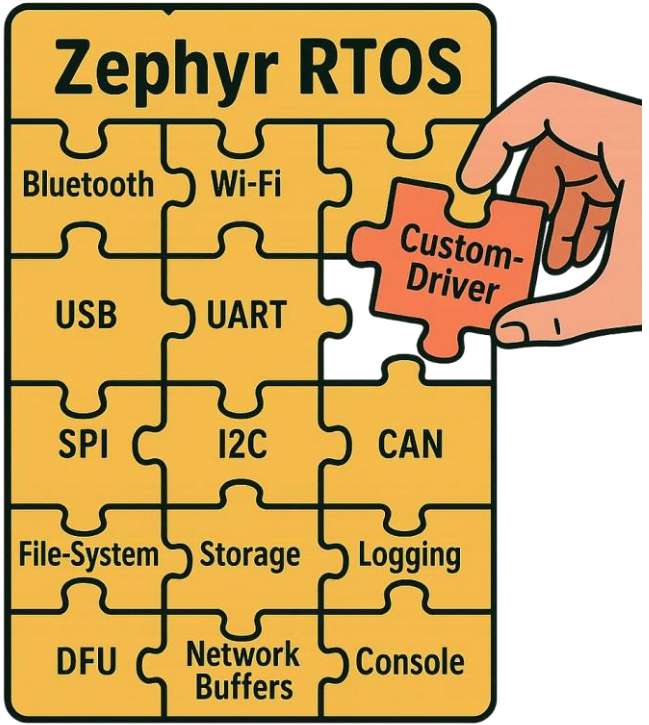
- Drivers may depend on other drivers being initialized first, or require the use of kernel services.
- Driver Initialization Levels:
  - PRE_KERNEL_1:
    - no dependencies, such as those that rely solely on hardware present in the processor/SOC.
    - cannot use any kernel services during configuration
  - PRE_KERNEL_2:
    - Based on PRE_KERNEL_1 level software
    - cannot use any kernel services during configuration
  - POST_KERNEL:
    - kernel services required during configuration
    - Init functions at this level run in context of the kernel main task.

Within each initialization level you specify a priority level.



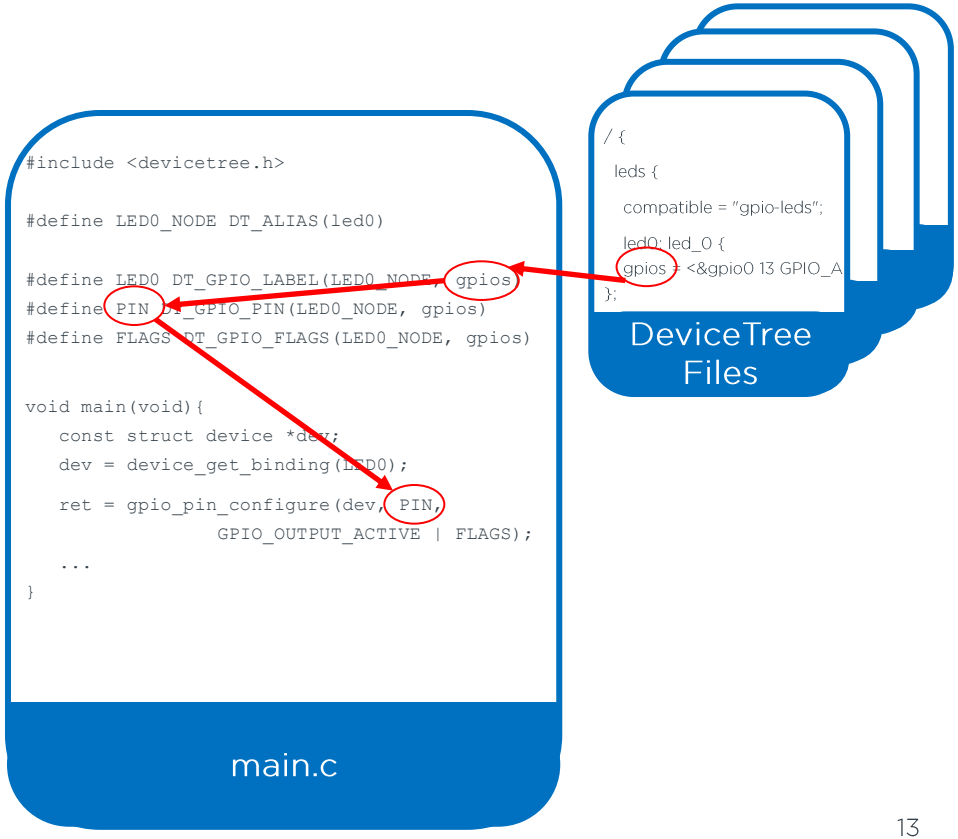
How to add a custom Driver?

- Where should I store the driver?
- How to add the Driver C Code to my project?
- How to Initialize/Start the Driver
- How to define Hardware Usage?



DeviceTree

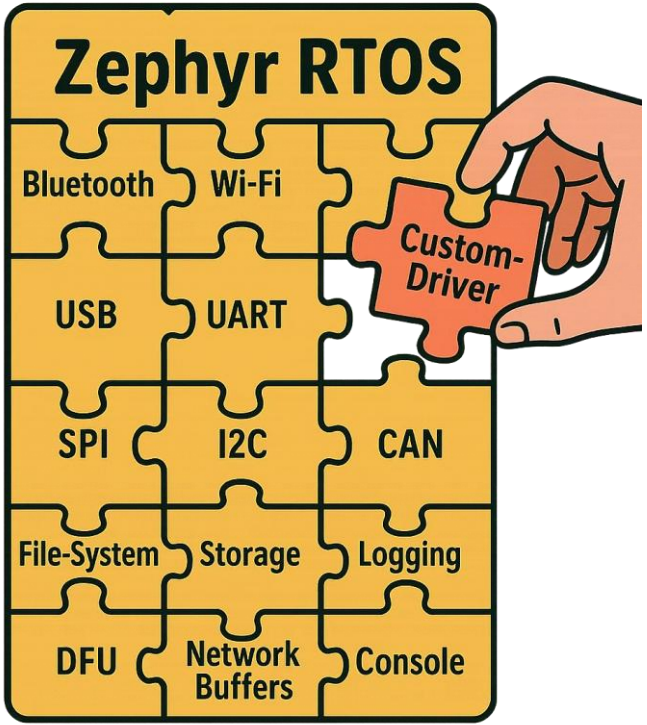
- Defining Hardware Usage in separate files (DeviceTree file)
- These Symbols are then used in your Code
- Advantage:
  - Can easily be reused (Hardware can then adjusted by just changing the DeviceTree file)
  - Perfect for Device Driver
  - No Memory overhead due to macros



How to add a custom Driver?

- Where should I store the driver?
- How to add the Driver C Code to my project?
- How to Initialize/Start the Driver
- How to define Hardware Usage?

- Power Management in the Driver
- …



Where can you find further information?

- Zephyr / nRF Connect SDK Documentation
- Nordic‘s Developer Academy (*nRF Connect SDK Fundamentals* course and *nRF Connect SDK Intermediate* course cover Drivers)
- Samples (Zephyr‘s sample folder => zephyr/samples and NRF Connect SDK sample folder => nrf/samples)

- Zephyr Developer Summit 2022:

  Tutorial: Mastering Zephyr Driver Development

DevAcademy

https://academy.nordicsemi.com/

Learning Paths in Nordic Developer Academy